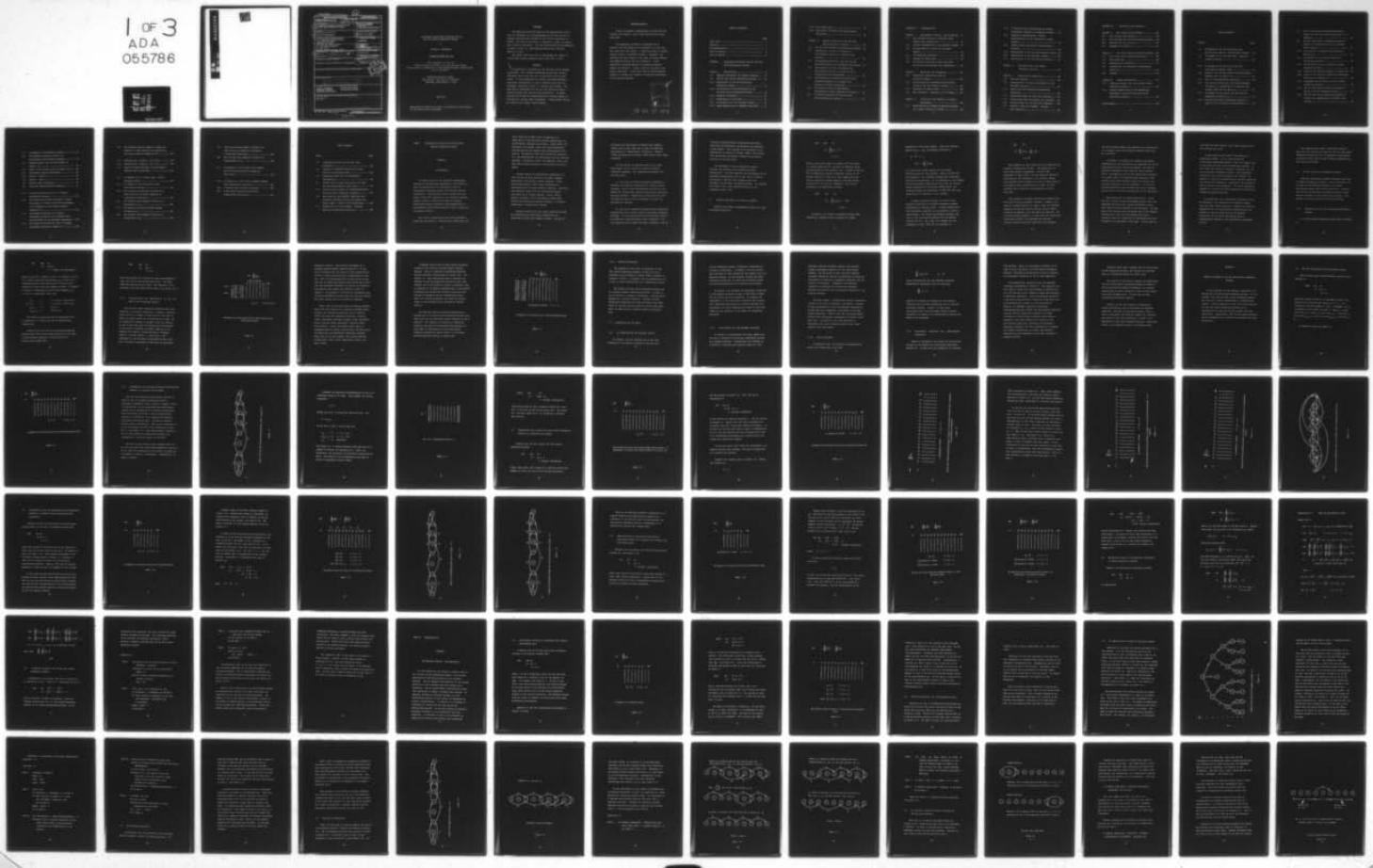
AD-A055 786 MASSACHUSETTS INST OF TECH CAMBRIDGE OPERATIONS RESE--ETC F/G 12/1
A LAGRANGEAN RELAXATION ALGORITHM FOR THE TWO DUTY PERIOD SCHED--ETC(U)
JUN 78 W B SHEPARDSON
UNCLASSIFIED TR-152 ARO-14261.9-M DAAG29-76-C-0064 NL

1 OF 3
ADA
055786



Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(12)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: ~~Technical Report No. 152~~
2. GOVT ACCESSION NO. (14) TR-152
3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle) (6): A LAGRANGEAN RELAXATION ALGORITHM FOR THE TWO DUTY PERIOD SCHEDULING PROBLEM.

5. TYPE OF REPORT & PERIOD COVERED (9): Technical Report, June 1978

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s) (10): Wilfred B. Shepardson

8. CONTRACT OR GRANT NUMBER(s) (15): DAAG29-76-C-0064

9. PERFORMING ORGANIZATION NAME AND ADDRESS
M.I.T. Operations Research Center
77 Massachusetts Avenue
Cambridge, MA 02139

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
~~P-14261-M~~

11. CONTROLLING OFFICE NAME AND ADDRESS
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709

12. REPORT DATE (11): June 1978

13. NUMBER OF PAGES: 236 pages (12) 238 p.

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) (18): ARO (19) 14261.9-M

15. SECURITY CLASS. (of this report): Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Releasable without limitation on dissemination.

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

DDC RECEIVED JUN 28 1978 F

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

THE VIEW, OPINIONS, AND/OR FINDINGS CONTAINED IN THIS REPORT ARE THOSE OF THE AUTHOR(S) AND SHOULD NOT BE CONSTRUED AS AN OFFICIAL DEPARTMENT OF THE ARMY POSITION, POLICY, OR DECISION, UNLESS SO DESIGNATED BY OTHER DOCUMENTATION.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Integer Programming
Lagrangean Techniques
Subgradient Optimization
Shortest Path Problems
Network Flow Problems

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

See page 2.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

270720

CL

A LAGRANGEAN RELAXATION ALGORITHM FOR THE TWO DUTY PERIOD SCHEDULING PROBLEM

by

WILFRED B. SHEPARDSON

Technical Report No. 152

Work Supported, in Part, by
Contract DAAG29-76-C-0064, U.S. Army Research Office
"Basic Studies in Combinatorial and Nondifferentiable Optimization"
M.I.T. OSP 84475

Operations Research Center
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

June 1978

FOREWORD

The Operations Research Center at the Massachusetts Institute of Technology is an interdepartmental activity devoted to graduate education and research in the field of operations research. The work of the Center is supported, in part, by government contracts and grants. The work reported herein was supported (in part) by the U.S. Army Research Office under Contract DAAG29-76-C-0064.

The author would also like to acknowledge the support of the National Science Foundation under Grant MCS 77-07327.

ABSTRACT

An algorithm is presented for the two duty period scheduling problem. This integer programming problem has a binary constraint matrix with two sets of consecutive ones in each column. At each subproblem of a branch and bound procedure, subgradient optimization is used to maximize the value of a Lagrangean relaxation, which is a network flow problem. The algorithm is implemented for the two duty period set partitioning problem, with shortest path relaxations. A second algorithm utilizing the unique properties of prime numbers is developed for solving small subproblems. Computational results are reported for several large problems.

ACKNOWLEDGEMENTS

I wish to express my appreciation to Julie for her devotion and support, and to Todd and Brett for being VERY good.

I am especially grateful to Professor Roy E. Marsten, who has guided me and assisted me in the work that went into this thesis. I would also like to express special thanks to Professor Thomas L. Magnanti, who directed the later stages of the work and whose comments and suggestions have been very valuable. I also appreciate the participation of Professors Jeremy F. Shapiro and Sanjoy K. Mitter, both of whom were on my thesis committee. I would also like to thank Professor Timothy L. Shaftel for a series of fruitful discussions on the Helsinki problem.

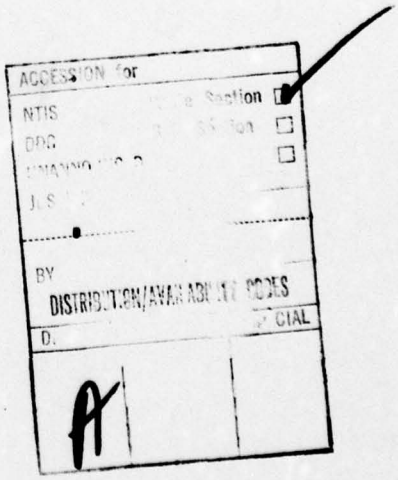


TABLE OF CONTENTS

SECTION II Implementation

SECTION III Extensions and Discussion

LIST OF FIGURES

LIST OF TABLES

PART I Lagrangean Relaxation and the Two Duty Period Scheduling Problem

CHAPTER 1

INTRODUCTION

In recent years we have witnessed considerable advances in mathematical programming. One productive area of concentration has concerned the use of Lagrangean relaxation coupled with creative problem formulation. Manifestations have included Held and Karp's (1970,1971) approach to the traveling salesman problem, Fisher's (1972,1973) study of resource constrained machine sequencing, and applications to multicommodity flow problems [Assad (1976), Kennington and Shalaby (1977)].

The notion of subgradients has also attracted a great deal of research. Held and Karp (1970,1971) and

Held, Wolfe and Crowder (1974) demonstrate its application to the traveling salesman problem and the multicommodity maximum flow problem. Assad (1976) and Kennington and Shalaby (1977) have used subgradient optimization for the minimal cost multicommodity flow problem. Etcheberry (1976) has utilized the technique in a new algorithm for set partitioning and set covering problems. Cornuejols, Fisher and Nemhauser (1977) have used this solution strategy to solve facility location models.

Another advance in mathematical programming has been the use of group theoretic and other knapsack equivalents of arbitrary integer programs. Fisher, Northup and Shapiro (1975) report computational experience with the group theoretic approach. Garfinkel and Nemhauser (1972) give an alternative knapsack equivalent of an integer program. Glover (1967), Shapiro and Wagner (1967) and Shapiro (1968) offer methods for solving the knapsack problem in a dynamic programming - shortest path context.

Another fruitful area for recent research has been the combination of algorithmic capabilities of operations research and computer science. The use of

efficient list processing techniques from computer science has in many cases led to order of magnitude improvements in computational efficiency. Magnanti (1976) and Magnanti and Golden (1978) survey many recent instances.

In this thesis we embellish upon all of these developments in the context of certain personnel scheduling problems. Our research has extended into two main areas:

1) the study of efficient dynamic programming - shortest path solution techniques for certain integer programs. Part of this investigation has led to a new method for transforming integer programs into equivalent knapsack problems. Computational experience has shown the effectiveness of these techniques for small set partitioning problems.

2) the use of Lagrangean relaxation and subgradient optimization for a certain class of personnel scheduling problems. The two duty period set partitioning problem we study has a natural shortest path relaxation. For each subproblem in a branch and bound strategy we use an

iterative backward-forward reaching shortest path algorithm and subgradient optimization for computing strong bounds. This approach is an outgrowth of an observation by Veinott and Wagner (1962) that certain very specialized scheduling problems can be solved directly as shortest paths.

The remainder of this introductory chapter outlines the problems to be studied and their applications. We first summarize the methodology of our knapsack transformation and its use within a dynamic programming - shortest path framework and then introduce the personnel scheduling problem. We conclude the chapter with a more detailed summary of the remainder of this work.

1.1 Knapsack Equivalents of Integer Programs

Consider any integer programming problem (IP) (with st meaning subject to)

$$\text{(IP)} \qquad \text{Min} \qquad CX$$

$$\text{st} \quad \sum_{j=1}^{N} A_j X_j = b$$

$$X \text{ in } S$$

where A_j and b are integer M-vectors in Z^M and where S, a given subset of Z^N, captures the discrete nature characteristics of the problem, e.g. S might be the set Z_+^N of nonnegative integer N-vectors, the set of 0,1 integer vectors, or some other more complicated set. Given a one-to-one linear mapping P from the integer M-vectors Z^M into the real numbers R, (IP) may be rewritten as an equivalent problem:

$$\text{(KIP)} \qquad \text{Min} \qquad CX$$

$$\text{st} \quad \sum_{j=1}^{N} P(A_j) X_j = P(b)$$

$$X \text{ in } S$$

In Chapter 5 we study the knapsack problem (KIP) defined by a mapping P which utilizes the unique

properties of the prime numbers. Given any distinct primes $P_1, P_2, \ldots, P_M$, the mapping P defined by

$$P: \quad Z^M \rightarrow R$$

$$P(z) = \sum_{i=1}^{M} z_i \ln P_i$$

is a one-to-one linear mapping of the integer M-vectors into the real numbers. (KIP), derived with this mapping P, is a knapsack problem with irrational constraint coefficients. If the variables X_j are bounded and the irrational coefficients $P(A_j), P(b)$ are closely enough approximated by rational numbers A_j^*, b^*, the resulting problem is equivalent to (KIP) and (IP) (see Proposition 5.1, Section 5.3 for a proof).

A common solution strategy for solving these knapsack equivalents of integer programs is dynamic programming or shortest paths. A useful property for this algorithmic approach is the property of optimal subsolutions. For integer programming problems (IP) with $S = Z_+^M$, any partial solution $\bar{X}$ of an optimal solution X^* (i.e. $\bar{X}_j \leq X_j^*$), $\bar{X}$ is optimal for the resources it uses. That is, $\bar{X}$ is optimal in

$$\text{Min} \quad CX$$

$$\text{st} \quad \sum_{j=1}^{N} A_j X_j = \sum_{j=1}^{N} A_j \bar{X}_j$$

$$X \text{ in } Z_+^M$$

This property is very useful when one considers the knapsack reformulation of (IP). The implication is, when using dynamic programming - shortest path techniques to solve (KIP), one may impose an arbitrary ordering upon the variables of the solution. Rather than consider every variable as an arc at every node of the shortest path interpretation of (KIP), we need consider only a subset of the variables as arcs at each node.

This property of optimal subsolutions depends upon the problem having unbounded variables. However, for some 0,1 integer problems the upper bounds are enforced naturally by the constraints (without explicit upper bounds on variables) or by the objective function. The set partitioning problem is a case where the constraints force each variable to be between zero and one. Multiple choice constraints also have this property.

The set covering problem with positive cost coefficients is an example where the objective function forces the 0,1 condition.

In Chapter 5 we exploit the property of optimal subsolutions in our development of algorithms for the set partitioning and set covering problems. We order the variables according to the first row with a nonzero entry. Consequently, for the set partitioning problem, at a given node of the shortest path problem we need consider only those variables whose first entry is in the first constraint not yet satisfied. For the set covering problem we need consider only those variables with an entry in the first constraint not yet satisfied.

These ideas have a broader application. In the case of a general integer programming problem (IP) with unbounded variables and a sparse coefficient matrix, there also is a good ordering of the variables. At each node of the shortest path interpretation of the knapsack formulation of (IP) one need consider only those variables which will contribute to satisfying the first constraint not satisfied at that node. In this way, at

each node one need consider only a small subset of the arcs leaving that node.

In this paper we will apply these ideas to a particular problem - the two duty period set partitioning scheduling problem, which is a special case of the class of K duty period scheduling problems. We will first use duality to develop a relaxation for the two duty period scheduling problem. This Lagrangean relaxation is an acyclic shortest path problem. We suggest an iterative backward-forward reaching algorithm for solving the relaxation. Subgradient optimization is used to maximize the Lagrangean. The entire process is embedded in a branch and bound strategy for solving the scheduling problem optimally.

For subproblems with dimensions of manageable size, we reformulate the subproblem as a knapsack problem using the prime number mapping P. Because the original subproblem is a set partitioning problem and has the property of optimal subsolutions, the knapsack formulation can be solved efficiently using a predetermined ordering of the variables, as mentioned earlier.

The computational results reported in Chapter 7 show that this solution strategy for the two duty period scheduling problem has led to significant improvements in solution times over the best existing algorithms for this class of problems.

1.2 The Two Duty Period Scheduling Problem

Personnel scheduling problems frequently give rise to integer programming problems. We now introduce the two duty period scheduling problem, an integer programming problem with special structure. A special case of the two duty period scheduling problem arises when the problem is a set partitioning problem. This special case will be called the Helsinki problem.

1.2.1 Formulation of the Two Duty Period Scheduling Problem

A two duty period scheduling problem may be defined as

$$(P) \qquad \text{Min} \quad CX$$
$$\text{st} \quad AX \lesseqgtr b$$
$$X \quad \text{Integer and Nonnegative}$$

where b is an M X 1 vector, C is a 1 X N vector, X is an N X 1 vector and where the entries, a_{ij}, of the M X N constraint matrix A are either zero or one and each column of A has at most two segments of ones. A SEGMENT of ones is a consecutive set of column elements a_{ij}, $i = k,k+1,\ldots,k+p-1,k+p$ such that

$$\begin{aligned} & a_{ij} = 1 && i = k,k+1,\ldots,k+p-1,k+p \\ \text{or} \quad & a_{ij} = -1 && i = k,k+1,\ldots,k+p-1,k+p \\ & a_{k-1,j} = 0 && \text{if } k > 1 \\ & a_{k+p+1,j} = 0 && \text{if } k+p < M \end{aligned}$$

The symbol $\lesseqgtr$ indicates that the constraints may be equalities or either less than or greater than inequalities.

A special case of the two duty period scheduling problem which will receive much attention in this work is the Helsinki problem or two duty period set partitioning problem (HP)

$$
\begin{aligned}
\text{(HP)} \qquad & \text{Min} \quad CX \\
& \text{st} \quad AX = 1 \\
& \qquad\quad X = 0,1
\end{aligned}
$$

where each column of A contains at most two segments of ones, the rest of the entries being zero. The right hand side elements are all ones. The notation X = 0,1 indicates that each entry of the vector X is either 0 or 1.

1.2.2 Interpretation and Applications of the Two Duty Period Scheduling Problem

The two duty period scheduling problem arises very naturally in personnel scheduling. Consider a situation where there are a number of jobs each of which must be done by a single person for a given period of time. As an illustrative example we consider buses and drivers. It was in this form that the problem was first brought to the author's attention by Markku Tamminen of the Helsinki Data Center, Finland [Koljonen & Tamminen (1977)]. Consider Figure 1.1 where rows 1 and 2 correspond to the two hours of operation of Bus A and rows 3 through 8 correspond to six hours of continuous

$$\text{Min} \quad \sum_{j=1}^{6} C_j X_j$$

Driver Schedules

		X_1	X_2	X_3	X_4	X_5	X_6		RHS
Bus A	8 am - 9 am	1	0	1	1	0	0	=	1
	9 am - 10 am	0	0	1	1	0	0	=	1
Bus B	9 am - 10 am	0	1	0	0	1	0	=	1
	10 am - 11 am	0	1	0	0	1	0	=	1
	11 am - 12 m	0	1	1	0	0	1	=	1
	12 m - 1 pm	1	0	1	1	0	1	=	1
	1 pm - 2 pm	1	0	0	1	1	0	=	1
	2 pm - 3 pm	0	1	0	0	1	0	=	1

$$X_j = 0,1 \qquad j = 1,2,3,4,5,6$$

An Example of the Two Duty Period Set Partitioning Problem

The Helsinki Problem

Figure 1.1

operation of Bus B. Each column corresponds to a possible driver schedule where an entry of 1 in the matrix indicates that the driver of that column drives the bus of that row for the hour corresponding to that row. With the restriction that a driver drive no more than one bus during his morning duty period and no more than one (possibly different) bus during his afternoon duty period, each column will contain at most two segments. The problem is to find a set of minimal cost of driver schedules such that each bus will have exactly one driver during each of its hours of operation.

Other cases of two duty period scheduling problems which have been cited in the literature include cyclic staffing with overtime [Bartholdi, Orlin & Ratliff (1977)] and days off scheduling [Brownell & Lowerre (1976), Tibrewala, Philippe & Browne (1972), Bartholdi, Orlin & Ratliff (1977), Bartholdi & Ratliff (1977), Orlin (1977)]. Glover and Mulvey (1976) refer to implementations of project partitioning, job processing, and monitoring and maintenance scheduling problems. Other authors who have studied very similar problems include Baker (1974, 1975), Maier-Rothe (1973), and Segal (1974).

A special case of the two duty period scheduling problem is the circular ones and cyclic staffing problem. This is a problem of scheduling personnel working continuous duty periods in cyclic time. See Figure 1.2. Each column represents a possible work schedule of one continuous duty period in real time. However, due to the arbitrary nature of starting a day at midnight in the matrix representation, a duty period beginning late one day and finishing early the next becomes two segments in the corresponding column. Efficient solution procedures are known for certain cases of this problem [Orlin, Bartholdi & Ratliff (1977)].

The two duty period scheduling problem can be generalized to a K duty period scheduling problem where each column of the constraint matrix contains at most K segments. For example, the problem of scheduling personnel who work five continuous duty periods in a week might be formulated as a five duty period scheduling problem in linear time or as a six duty period scheduling problem in cyclic time.

$$\text{Min} \quad \sum_{j=1}^{7} c_j x_j$$

	X_1	X_2	X_3	X_4	X_5	X_6	X_7		RHS
12 pm - 2 am	0	0	1	1	0	1	0	=	3
2 am - 4 am	0	1	1	1	0	0	0	=	2
4 am - 6 am	0	1	1	0	0	0	0	=	1
6 am - 8 am	0	1	1	0	0	0	0	=	1
8 am - 10 am	0	1	0	0	1	0	0	=	2
10 am - 12 m	0	1	0	0	1	0	0	=	1
12 m - 2 pm	0	0	0	0	1	0	1	=	2
2 pm - 4 pm	0	0	0	0	1	0	1	=	1
4 pm - 6 pm	1	0	0	0	0	0	1	=	1
6 pm - 8 pm	1	0	0	0	0	0	1	=	1
8 pm - 10 pm	1	0	1	0	0	0	0	=	1
10 pm - 12 pm	1	0	1	1	0	0	0	=	2

X_j Nonnegative, Integer $\quad j = 1,\ldots,7$

An Example of the Circular Ones - Cyclical Staffing Problem

Figure 1.2

1.2.3 Solution Techniques

The simplest of this class of problems, the one duty period scheduling problem, is known to have a unimodular matrix [Veinott & Wagner (1962), Garfinkel & Nemhauser (1972)]. In Chapter 2 we will demonstrate how it may be easily reformulated as a network flow problem.

The general two duty period scheduling problem (P) is a general integer programming problem. As such it can be solved by a number of techniques - cutting plane, enumeration, or group theoretic. However, no good specialized algorithms have been developed which might solve the two duty period problem in better times than might be expected from a general integer programming code.

1.3 Organization of the Paper

1.3.1 An Algorithm for the Helsinki Problem

In Chapter 2 we will indicate how we can take advantage of the special structure of the two duty

period scheduling problem to develop an algorithm for solving it efficiently. In Chapter 3 we will develop this algorithm in great detail for the special case, the Helsinki problem. We will develop a branch and bound enumeration technique, in which the relaxation solved for each subproblem will be a shortest path problem.

In Chapter 4 we introduce the Lagrangean relaxation of the Helsinki problem in order to get tighter bounds for our branch and bound procedure. To evaluate the Lagrangean, we need only solve a shortest path problem. When subgradient optimization is used to maximize the Lagrangean, the extra computational burden is more than offset by the reduction in the number of subproblems generated.

1.3.2 Prime Numbers and the HELSINKI Algorithm

In Chapter 5 we demonstrate how prime numbers can be used to reformulate any integer programming problem as a knapsack problem. Interpreting this knapsack is problem as a shortest path problem, leads to a very

efficient solution procedure whenever the original integer programming problem is a set partitioning problem. The existence of this efficient solution technique depends on the use of logarithms of primes as multipliers in obtaining a surrogate constraint for the original constraints. A somewhat less efficient algorithm is developed for solving the resulting shortest path problem when the original problem is a set covering problem.

The prime number - shortest path method is severely limited in the size of problem it can handle. However, we have incorporated it into the HELSINKI algorithm to be used when the subproblem to be solved is of small enough dimensions. Used in this way, the prime number - shortest path technique is guaranteed to fathom a node when used. We found it to be very effective in eliminating the need to separate nodes at the lower levels of the tree search.

1.3.3 Side Constraints

In Chapters 5 and 7 we develop two procedures by which side constraints of the form

$$\sum_{j=1}^{N} t_j X_j \leq T \qquad t_j \leq 0$$

can be incorporated into the HELSINKI algorithm. Computational experience with the constraint

$$\sum_{j=1}^{N} X_j \leq K$$

limiting the number of variables in the solution, indicates that the side constraints tend to slow the algorithm slightly. However, in some cases, particularly where the constraint cannot be easily satisfied, its effect is to significantly increase the speed of the algorithm.

1.3.4 Extensions, Discussion and Computational Experience

Chapter 6 generalizes the results of the previous chapters to the general two duty period scheduling problem (P). We show that the relaxation is a network

flow problem. Again, an enumerative procedure can be used to solve the general two duty period scheduling problem. Subgradient optimization is used to maximize the Lagrangean relaxation of (P) at each subproblem.

Our computational experience with the HELSINKI algorithm is presented in Chapter 7. Our algorithm has been tested on four real world probles of average size 65 X 475 (as well as on a number of small test problems artificially generated). Since the Helsinki problem is a set partitioning problem, specialized algorithms have been developed for solving it quite efficiently. However, the HELSINKI algorithm consistently outperformed Marsten's SETPAR set partitioning algorithm [Marsten (1971)], generally regarded as the best available set partitioning algorithm. Tests done by Tamminen on one test problem with a number of set partitioning algorithms (including the Garfinkel & Nemhauser algorithm) led that investigator to conclude that SETPAR outperformed all others (the HELSINKI algorithm was not developed at that time) [Koljonen & Tamminen (1977)].

Chapter 8 deals with a special case of the two duty period scheduling problem, the circular ones problem. This is a scheduling problem in cyclic time.

The results of the earlier chapters are generalized for the K duty period scheduling problem in Chapter 9. This K duty period problem corresponds to an integer program with at most K segments of ones in each column, other entries being zero. We show that the same theoretical development applies.

Finally, in the last chapter, we examine the general methodology used in developing the HELSINKI algorithm. The idea of decoupling columns, which is used to reformulate the Helsinki problem as a shortest path problem with side constraints, is shown to be applicable to near block diagonal matrices. There also exists a corresponding concept of decoupling rows.

CHAPTER 2

SPECIAL STRUCTURE OF THE TWO DUTY PERIOD SCHEDULING PROBLEM

In this chapter we will develop a methodology for solving the two duty period scheduling problem. We will consider first the one duty period scheduling problem and show how it may be transformed to a network flow problem. Then in Sections 2.4 and 2.5 we will show how the two duty period scheduling problem may be reformulated as a one duty period problem with side constraints. Equivalently, the two duty period problem may be reformulated as a network flow problem with side constraints.

2.1 The One Duty Period Set Partitioning Problem

The one duty period set partitioning problem may be defined as

$$
\begin{aligned}
\text{(1HP)} \quad & \text{Min} \quad CX \\
& \text{st} \quad AX = 1 \\
& \qquad X = 0,1
\end{aligned}
$$

where the entries of the M X N constraint matrix A are either zero or one and each column A_j, j = 1,...,N, contains at most one segment of ones. A SEGMENT S of A_j is defined to be a subset of rows, S = {p,p+1,...,p+k}, such that $a_{ij} = 1$ for all i in S (or $a_{ij} = -1$ for all i in S) and $a_{p-1,j} = 0$ if $p \neq 1$ and $a_{p+k+1,j} = 0$ if $p+k \neq M$.

An example is given in Figure 2.1.

$$\text{Min} \quad \sum_{j=1}^{11} c_j X_j$$

st

X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	X_{11}		RHS
1	0	0	0	1	0	1	0	0	0	0	=	1
0	0	0	0	1	0	1	0	0	0	0	=	1
0	0	1	0	0	0	0	0	1	0	0	=	1
0	0	1	0	0	0	0	0	1	0	0	=	1
0	0	1	0	0	1	0	0	0	0	1	=	1
0	1	0	0	0	1	0	1	0	0	1	=	1
0	1	0	0	0	0	0	1	0	1	0	=	1
0	0	0	1	0	0	0	0	0	1	0	=	1

$$X_j = 0,1 \qquad j = 1,2,\ldots,11$$

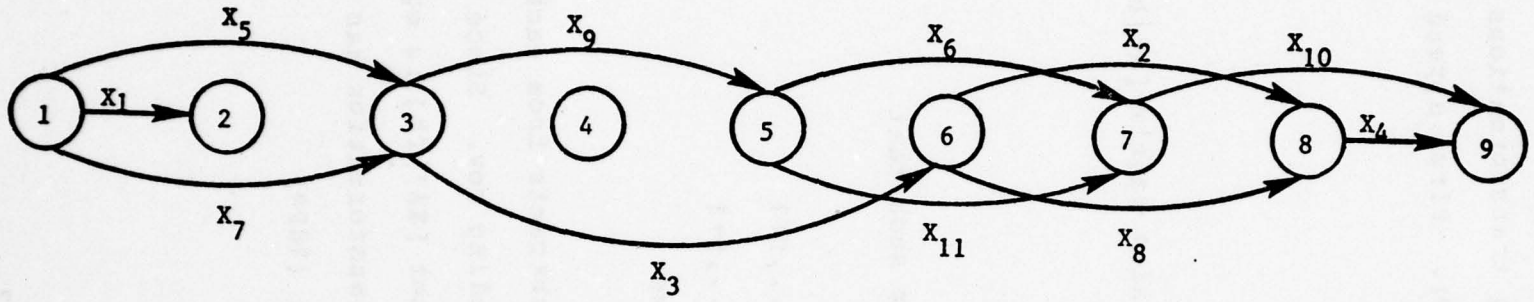
An Example of the One Duty Period Set Partitioning Problem (1HP)

Figure 2.1

2.2 Transforming the One Duty Period Set Partitioning Problem to a Shortest Path Problem

The one duty period set partitioning problem is known to have a unimodular constraint matrix A [Garfinkel & Nemhauser (1972), Veinott & Wagner (1962)]. It follows that the one duty period set partitioning problem can be represented as a shortest path problem with (M+1) nodes and N arcs. The set partitioning problem is to choose a set of columns that together cover every row exactly once. Consider a segment S covering rows p through p+k. This may be represented by an arc originating at node p and terminating at node p+k+1. See Figure 2.2. Then each feasible solution to the one duty period set partitioning problem will correspond to a path from node 1 to node M+1.

Each arc of this shortest path interpretation of the one duty period set partitioning problem is directed in the sense of originating at the lowest row number in the segment to which it corresponds. Consequently, the graph is acyclic.



Graph of the One Duty Period Scheduling Problem (1HP) of Figure 2.1

Figure 2.2

Consider the following transformations on the M X N constraint matrix A of (1HP). First append the trivial constraint

$$0 = 0$$

giving the (M+1) X N constraint matrix [A'|b]. Let

$$T = (t_{ij})$$

be the (M+1) X (M+1) matrix such that

$$t_{ii} = 1, \quad i = 1,\ldots,M+1$$
$$t_{i-1,j} = -1, \quad i = 2,\ldots,M+1$$
$$t_{ij} = 0, \quad \text{otherwise}$$

See Figure 2.3. T simply subtracts from each row of A' (except the first) the preceding row. Since T is nonsingular the constraint set [TA'|Tb] is equivalent to [A|1]. The effect of the transformation has been to create an equivalent problem (1HP*)

$$T_9 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix}$$

The 9 X 9 Transformation Matrix T_9

Figure 2.3

$$(1HP^*) \qquad \begin{aligned} \text{Min} \quad & CX \\ \text{st} \quad & TA'X = T1 \\ & X \ \text{Integer, Nonnegative} \end{aligned}$$

where each column of (TA') contains exactly one 1 and one -1, the rest of the entries being zero. See Figure 2.4. In other words TA'X = T1 represents a shortest path problem.

2.3 Transforming the General One Duty Period Scheduling Problem to a Network Flow Problem

Consider now the more general one duty period scheduling problem.

$$(1P) \qquad \begin{aligned} \text{Min} \quad & CX \\ \text{st} \quad & AX \geq b \\ & X \ \text{Integer, Nonnegative} \end{aligned}$$

where, once again, each column of A contains exactly one segment of ones, the rest of the entries being zero.

$$\text{Min} \quad \sum_{j=1}^{11} c_j X_j$$

st	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	X_{11}		RHS
	1	0	0	0	1	0	1	0	0	0	0	=	1
	-1	0	0	0	0	0	0	0	0	0	0	=	0
	0	0	1	0	-1	0	-1	0	1	0	0	=	0
	0	0	0	0	0	0	0	0	0	0	0	=	0
	0	0	0	0	0	1	0	0	-1	0	1	=	0
	0	1	-1	0	0	0	0	1	0	0	0	=	0
	0	0	0	0	0	-1	0	0	0	1	-1	=	0
	0	-1	0	1	0	0	0	-1	0	0	0	=	0
	0	0	0	-1	0	0	0	0	0	-1	0	=	-1

$$X_j = 0,1 \qquad j = 1,2,\ldots,11$$

The One Duty Period Set Partitioning Problem (1HP) of Figure 2.1 Transformed to a Shortest Path Problem (1HP*) by the use of T_9.

Figure 2.4

See the example in Figure 2.5. Then (1P) can be transformed to

$$
\begin{aligned}
(1P) \quad & \text{Min } CX \\
& \text{st } AX - IY = b \\
& \qquad X \text{ Integer, Nonnegative}
\end{aligned}
$$

by the addition of surplus variables Y. See the example in Figure 2.6. Notice that (1P) still satisfies the criterion that each column have exactly one segment. It is also the case for less than or equal to inequalities, $AX \leq b$, that the addition of slack variables will lead to a reformulation satisfying the criterion that each column have exactly one segment.

In the same manner that (1HP) was transformed to an acyclic shortest path problem, (1P) may be transformed to a network flow problem.

Consider the example given in Figure 2.6. Adding the trivial row

$$0 = 0$$

$$\text{Min} \quad \sum_{j=1}^{11} C_j X_j$$

st

X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	X_{11}		RHS
1	0	0	0	1	0	1	0	0	0	0	$\geq$	3
0	0	0	0	1	0	1	0	0	0	0	$\geq$	5
0	0	1	0	0	0	0	0	1	0	0	$\geq$	4
0	0	1	0	0	0	0	0	1	0	0	$\geq$	7
0	0	1	0	0	1	0	0	0	0	1	$\geq$	6
0	1	0	0	0	1	0	1	0	0	1	$\geq$	4
0	1	0	0	0	0	0	1	0	1	0	$\geq$	8
0	0	0	1	0	0	0	0	0	1	0	$\geq$	4

X_j Nonnegative, Integer $\quad j = 1,2,\ldots,11$

An Example of the General One Duty Period Scheduling Problem (1P)

Figure 2.5

Min $\sum_{j=1}^{11} c_j X_j$

st	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	X_{11}	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7	Y_8		RHS
	1	0	0	0	1	0	1	0	0	0	0	-1	0	0	0	0	0	0	0	=	3
	0	0	0	0	1	0	1	0	0	0	0	0	-1	0	0	0	0	0	0	=	5
	0	0	1	0	0	0	0	0	1	0	0	0	0	-1	0	0	0	0	0	=	4
	0	0	1	0	0	0	0	0	1	0	0	0	0	0	-1	0	0	0	0	=	7
	0	0	1	0	0	1	0	0	0	0	1	0	0	0	0	-1	0	0	0	=	6
	0	1	0	0	0	1	0	1	0	0	1	0	0	0	0	0	-1	0	0	=	4
	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	-1	0	=	8
	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	=	4

X_j Nonnegative, Integer $j = 1,2,\ldots,11$

The One Duty Period Scheduling Problem of Figure 2.5 with Surplus Variables Used to Transform the Constraints to Equations

Figure 2.6

gives the matrix in Figure 2.7. Then, after applying the transformation T, one gets the resulting matrix depicted in Figure 2.8. Our one duty period scheduling problem has been transformed to a network flow problem.

In the one duty period set partitioning problem (1HP) one unit of flow is to pass through a path from node 1 to node M+1. In the more general problem (1P) we must have b_i units of flow passing through each node i. Consequently, at each node we may have to introduce or withdraw units of flow. Precisely, we must introduce (or withdraw) $b_i - b_{i-1}$ units of flow at node i, where $b_0 = 0$. Now, we must add a source (Node 0) as well as a sink (Node M + 2). See Figure 2.9, where arcs originating at node 0 introduce flow to nodes and arcs ending at node 10 withdraw flow from nodes. If the constraints of (1P) had all been equations or less than or equal to inequalities, then the corresponding network flow representation would have been acyclic. As it is, each variable Y_i becomes an arc from node i + 1 to node i.

$$\text{Min} \quad \sum_{j=1}^{11} C_j X_j$$

st	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	X_{11}	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7	Y_8		RHS
	1	0	0	0	1	0	1	0	0	0	0	-1	0	0	0	0	0	0	0	=	3
	0	0	0	0	1	0	1	0	0	0	0	0	-1	0	0	0	0	0	0	=	5
	0	0	1	0	0	0	0	0	1	0	0	0	0	-1	0	0	0	0	0	=	4
	0	0	1	0	0	0	0	0	1	0	0	0	0	0	-1	0	0	0	0	=	7
	0	0	1	0	0	1	0	0	0	0	1	0	0	0	0	-1	0	0	0	=	6
	0	1	0	0	0	1	0	1	0	0	1	0	0	0	0	0	-1	0	0	=	4
	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	-1	0	=	8
	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	=	4
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	=	0

X_j Nonnegative, Integer $j = 1,2,\ldots,11$

The One Duty Period Scheduling Problem of Figure 2.6 with the Trivial Constraint 0 = 0 Appended

Figure 2.7

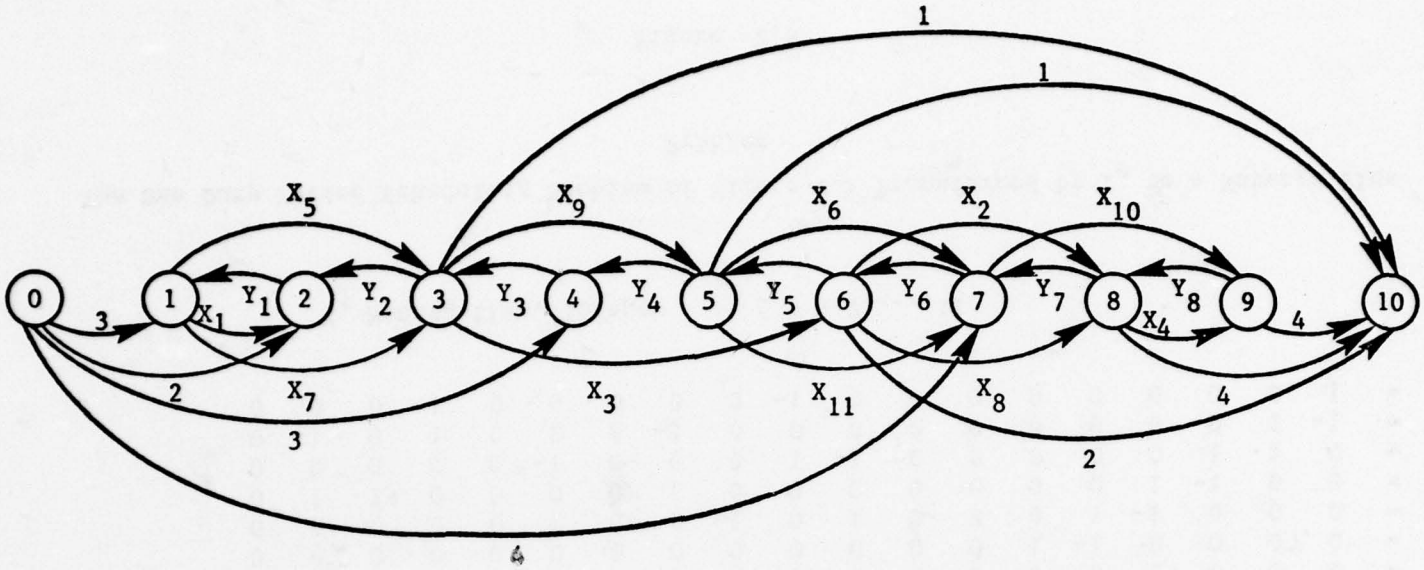
$$\text{Min} \quad \sum_{j=1}^{11} C_j X_j$$

st	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	X_{11}	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7	Y_8		RHS
	1	0	0	0	1	0	1	0	0	0	0	-1	0	0	0	0	0	0	0	=	3
	-1	0	0	0	0	0	0	0	0	0	0	1	-1	0	0	0	0	0	0	=	2
	0	0	1	0	-1	0	-1	0	1	0	0	0	1	-1	0	0	0	0	0	=	-1
	0	0	0	0	0	0	0	0	0	0	0	0	0	1	-1	0	0	0	0	=	3
	0	0	0	0	0	1	0	0	-1	0	1	0	0	0	1	-1	0	0	0	=	-1
	0	1	-1	0	0	0	0	1	0	0	0	0	0	0	0	1	-1	0	0	=	-2
	0	0	0	0	0	-1	0	0	0	1	-1	0	0	0	0	0	1	-1	0	=	4
	0	-1	0	1	0	0	0	-1	0	0	0	0	0	0	0	0	0	1	-1	=	-4
	0	0	0	-1	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	=	-4

X_j Nonnegative, Integer $j = 1,2,\ldots,11$

The One Duty Period Scheduling Problem of Figure 2.5 Transformed by T_9 to a Network Flow Problem

Figure 2.8



Labels on arcs originating at node 0 or ending at node 10 indicate amount of flow.

Figure 2.9

2.4 Reformulating the Two Duty Period Set Partitioning Problem as a Shortest Path Problem with Side Constraints

Consider now the two duty period set partitioning problem which we will call the Helsinki problem (HP)

$$
\begin{aligned}
\text{(HP)} \quad & \text{Min } CX \\
& \text{st } AX = 1 \\
& \quad X = 0,1
\end{aligned}
$$

where each column of A contains one or two segments of ones, the rest of the entries being zero. An example is given in Figure 2.10. This example corresponds to the bus driver example given in Figure 1.1 in Chapter 1. (HP) and the example in Figure 2.10 are both set partitioning problems. However, both have the special property of having at most two segments in each column.

We know that the one duty period set partitioning problem is easily solved. What implications does this have for the two duty period set partitioning problem? The shortest path interpretation of the single segment set partitioning problem suggests a natural relaxation for the two segment problem.

$$\text{Min} \quad \sum_{j=1}^{6} c_j X_j$$

st	X_1	X_2	X_3	X_4	X_5	X_6		RHS
	1	0	1	1	0	0	=	1
	0	0	1	1	0	0	=	1
	0	1	0	0	1	0	=	1
	0	1	0	0	1	1	=	1
	0	1	1	0	0	1	=	1
	1	0	1	1	0	1	=	1
	1	0	0	1	1	0	=	1
	0	1	0	0	1	0	=	1

$$X_j = 0,1 \quad j = 1,2,\ldots,6$$

An Example of the Two Duty Period Set Partitioning Problem

Figure 2.10

Consider again the Helsinki problem example in Figure 2.10. Suppose each column A_j containing two segments were separated into two columns, A_j^1 and A_j^2, each containing one segment, see Figure 2.11. The matrix in Figure 2.11 corresponds exactly to that of Figure 2.1.

In terms of our shortest path interpretation, each variable X_j of the original problem corresponds to two arcs, X_j^1 and X_j^2, (see Figure 2.12). Dividing the columns decouples the arcs (see Figure 2.13). If we impose the constraints that each arc must have the same value as its partner (i.e. $X_j^1 = X_j^2$, $j = 1,\ldots,N$) then our new problem (HP') is equivalent to the original problem. Proposition 2.1 formalizes this idea in Section 2.6.

$$
\begin{array}{lll}
\text{(HP')} & \text{Min} & C^1 X^1 + C^2 X^2 \\
 & \text{st} & A^1 X^1 + A^2 X^2 = 1 \\
 & & IX^1 - IX^2 = 0 \\
 & & X^1, X^2 = 0,1
\end{array}
$$

where $C^1 + C^2 = C$.

$$\text{Min} \quad \sum_{j=1}^{6} C_j^1 X_j^1 \quad + \quad \sum_{j=1}^{5} C_j^2 X_j^2$$

	X_1		X_2		X_3		X_4		X_5				
st	X_1^1	X_1^2	X_2^1	X_2^2	X_3^1	X_3^2	X_4^1	X_4^2	X_5^1	X_5^2	X_6^1		RHS
	1	0	0	0	1	0	1	0	0	0	0	=	1
	0	0	0	0	1	0	1	0	0	0	0	=	1
	0	0	1	0	0	0	0	0	1	0	0	=	1
	0	0	1	0	0	0	0	0	1	0	0	=	1
	0	0	1	0	0	1	0	0	0	0	1	=	1
	0	1	0	0	0	1	0	1	0	0	1	=	1
	0	1	0	0	0	0	0	1	0	1	0	=	1
	0	0	0	1	0	0	0	0	0	1	0	=	1

$$X_j^1 = X_j^2 \qquad j = 1,2,\ldots,5$$

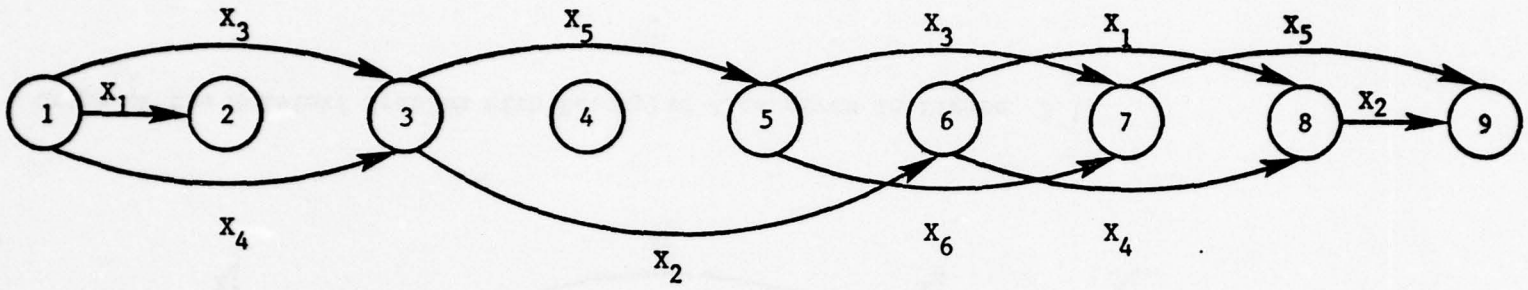
$$X_j^1 = 0,1 \qquad j = 1,2,\ldots,6$$

$$X_j^2 = 0,1 \qquad j = 1,2,\ldots,5$$

$$\text{where } C_j^1 + C_j^2 = C_j \qquad j = 1,2,\ldots,5$$

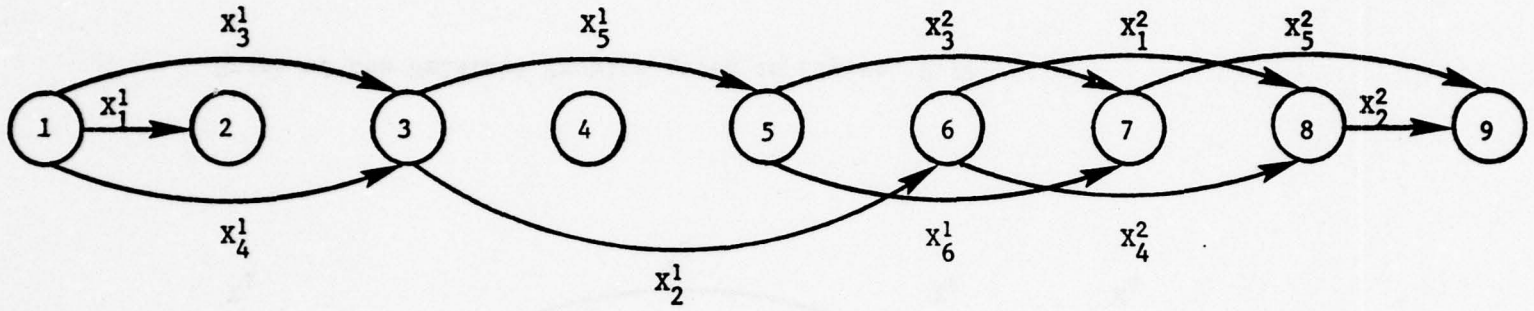
The Helsinki Problem of Figure 2.10 with Decoupled Columns

Figure 2.11



Graph of the Helsinki Problem given in Figure 2.10

Figure 2.12



Graph of the Helsinki Problem with Decoupled Arcs given in Figure 2.11

Figure 2.13

That is, the Helsinki problem is equivalent to an acyclic shortest path problem with coupled arcs. Similarly, it can be shown that the more general two duty period scheduling problem is equivalent to a network flow problem with coupled arcs.

2.5 Reformulating the General Two Duty Period Scheduling Problem as a Network Flow Problem with Side Constraints

Consider now the general two duty period scheduling problem (P) (see Figure 2.14)

$$
\begin{aligned}
(P) \quad & \text{Min} && CX \\
& \text{st} && AX = b \\
& && X \ \text{Integer, Nonnegative}
\end{aligned}
$$

where each column of A contains at most two segments of ones, other entries being zero. Inequalities in the original constraints may be transformed to equations by the use of surplus and slack variables.

$$\text{Min} \quad \sum_{j=1}^{6} C_j X_j$$

st	X_1	X_2	X_3	X_4	X_5	X_6		RHS
	1	0	1	1	0	0	=	3
	0	0	1	1	0	0	=	5
	0	1	0	0	1	0	=	4
	0	1	0	0	1	1	=	7
	0	1	1	0	0	1	=	6
	1	0	1	1	0	1	=	4
	1	0	0	1	1	0	=	8
	0	1	0	0	1	0	=	4

X_j Nonnegative, Integer $j = 1,2,\ldots,6$

An Example of the General Two Duty Period Scheduling Problem

Figure 2.14

Replace each variable X_j with two variables X_j^1 and X_j^2. Partition the non-zero entries of the A matrix into two matrices A^1 and A^2 where A^1 represents the first segments of the variables and A^2 represents the second segments (where they exist). $A = A^1 + A^2$. Divide the costs C into C^1 and C^2 where $C = C^1 + C^2$. The new problem then is given by (P') (see Figure 2.15)

$$\begin{aligned} (P') \quad \text{Min} \quad & C^1X^1 + C^2X^2 \\ \text{st} \quad & A^1X^1 + A^2X^2 = b \\ & X^1, X^2 \quad \text{Integer, Nonnegative} \end{aligned}$$

where $C^1 + C^2 = C$

To the constraints $[A^1|A^2|b]$, append the trivial constraint

$$0 = 0$$

to get a new constraint matrix $[A^{1'}|A^{2'}|b']$. Then apply transformation T to get $[TA^{1'}|TA^{2'}|Tb']$. See Figure 2.16. Then $[TA^{1'}|TA^{2'}|Tb']$ can be interpreted as a network flow problem. And our reformulation of (P)

$$\text{Min} \quad \sum_{j=1}^{6} C_j^1 X_j^1 + \sum_{j=1}^{5} C_j^2 X_j^2$$

st	X_1^1	X_1^2	X_2^1	X_2^2	X_3^1	X_3^2	X_4^1	X_4^2	X_5^1	X_5^2	X_6^1		RHS
	1	0	0	0	1	0	1	0	0	0	0	=	3
	0	0	0	0	1	0	1	0	0	0	0	=	5
	0	0	1	0	0	0	0	0	1	0	0	=	4
	0	0	1	0	0	0	0	0	1	0	0	=	7
	0	0	1	0	0	1	0	0	0	0	1	=	6
	0	1	0	0	0	1	0	1	0	0	1	=	4
	0	1	0	0	0	0	0	1	0	1	0	=	8
	0	0	0	1	0	0	0	0	0	1	0	=	4

$$X_j^1 = X_j^2 \qquad j = 1,2,\ldots,5$$

X_j^1 Nonnegative, Integer $\qquad j = 1,2,\ldots,6$

X_j^2 Nonnegative, Integer $\qquad j = 1,2,\ldots,5$

The Two Duty Period Scheduling Problem of Figure 2.14 with Decoupled Columns

Figure 2.15

$$\text{Min} \quad \sum_{j=1}^{6} C_j^1 X_j^1 + \sum_{j=1}^{5} C_j^2 X_j^2$$

st

X_1^1	X_1^2	X_2^1	X_2^2	X_3^1	X_3^2	X_4^1	X_4^2	X_5^1	X_5^2	X_6^1		RHS
1	0	0	0	1	0	1	0	0	0	0	=	3
-1	0	0	0	0	0	0	0	0	0	0	=	2
0	0	1	0	-1	0	-1	0	1	0	0	=	-1
0	0	0	0	0	0	0	0	0	0	0	=	3
0	0	0	0	0	1	0	0	-1	0	1	=	-1
0	1	-1	0	0	0	0	1	0	0	0	=	-2
0	0	0	0	0	-1	0	0	0	1	-1	=	4
0	-1	0	1	0	0	0	-1	0	0	0	=	-4
0	0	0	-1	0	0	0	0	0	-1	0	=	-4

$$X_j^1 = X_j^2 \qquad j = 1,2,\ldots,5$$

X_j^1 Nonnegative, Integer $\quad j = 1,2,\ldots,6$

X_j^2 Nonnegative, Integer $\quad j = 1,2,\ldots,5$

The Two Duty Period Scheduling Problem of Figure 2.14 Transformed to a Network Flow Problem

Figure 2.16

$$
\begin{aligned}
(P^*) \qquad \text{Min} \quad & C^1X^1 + C^2X^2 \\
\text{st} \quad & TA^1{}'X^1 + TA^2{}'X^2 = Tb' \\
& IX^1 - IX^2 = 0 \\
& X^1, X^2 \text{ Integer, Nonnegative}
\end{aligned}
$$

can be interpreted as a network flow problem with side constraints. The nature of these side constraints is to couple pairs of variables, forcing each pair to have the same value. Hence, the two duty period scheduling problem is equivalent to a network flow problem with coupled arcs.

2.6 Decoupling Columns in Mathematical Programming to Obtain Equivalent Problems

Consider the mathematical programming problem

$$
\begin{aligned}
(LP) \qquad \text{Min} \quad & CX \\
\text{st} \quad & AX = b
\end{aligned}
$$

or equivalently

$$\text{(LP)} \qquad \text{Min} \quad \sum_{j=1}^{N} C_j X_j$$

$$\text{st} \quad \sum_{j=1}^{N} A_j X_j = b$$

where A_j is the jth column of the MxN matrix A. Suppose each vector $[C_j | A_j]$ were to be replaced by k_j vectors

$$[C_j^k | A_j^k] \qquad k = 1,2,\ldots,k_j$$

having the property that

$$[C_j | A_j] = \sum_{j=1}^{k_j} [C_j^k | A_j^k], \qquad j = 1,\ldots,N$$

and simultaneously X_j is replaced by $[X_j^1,\ldots,X_j^{k_j}]$. We can then create a new problem (LP*) with these new variables plus the new constraints $X_j^k = X_j^{k+1}$, $k = 1,\ldots,k_j-1$, $j = 1,\ldots N$.

$$\text{(LP*)} \qquad \text{Min} \quad \sum_{j=1}^{N} \sum_{k=1}^{k_j} C_j^k X_j^k$$

$$\text{st} \quad \sum_{j=1}^{N} \sum_{k=1}^{k_j} A_j^k X_j^k = b$$

$$X_j^k - X_j^{k+1} = 0, \quad k = 1,\ldots,k_j - 1$$

$$j = 1,\ldots,N$$

Proposition 2.1 (LP*) is equivalent to (LP).

Proof: Part 1

Let $X = (X_1, X_2, \ldots, X_N)$ be a solution to (LP)

let $X_j^{k*} = X_j \qquad j = 1,2,\ldots,N \qquad k = 1,\ldots,k_j$

then $X_j^{k*} = X_j^{k+1*}$, $k = 1,\ldots,k_j - 1$; $j = 1,2,\ldots,N$

and $$\sum_{j=1}^{N} \sum_{k=1}^{k_j} A_j^k X_j^{k*} = \sum_{j=1}^{N} \sum_{k=1}^{k_j} A_j^k X_j = \sum_{j=1}^{N} A_j X_j = b$$

and $$\sum_{j=1}^{N} \sum_{k=1}^{k_j} C_j^k X_j^{k*} = \sum_{j=1}^{N} \sum_{k=1}^{k_j} C_j^k X_j = \sum_{j=1}^{N} C_j X_j = CX$$

$\Rightarrow$ $X^* = (X_1^{1*}, X_1^{2*}, \ldots, X_1^{k_1*}, X_2^{1*}, \ldots, X_N^{k_N*})$ is a solution to (LP*) with value CX

Part 2

Let $X^* = (X_1^{1*}, \ldots, X_1^{k_1*}, \ldots, X_N^{k_N*})$ be a solution to (LP*)

then $X_j^{1*} = X_j^{2*} = \ldots = X_j^{k_j*}$, $j = 1,2,\ldots,N$

let $X_j = X_j^{1*} \qquad j = 1,\ldots,N$

$$\text{then} \quad \sum_{j=1}^{N} C_j X_j = \sum_{j=1}^{N} \left(\sum_{k=1}^{k_j} C_j^k \right) X_j = \sum_{j=1}^{N} \sum_{k=1}^{k_j} C_j^k X_j^{k*}$$

$$\text{and} \quad \sum_{j=1}^{N} A_j X_j = \sum_{j=1}^{N} \left(\sum_{k=1}^{k_j} A_j^k \right) X_j = \sum_{j=1}^{N} \sum_{k=1}^{k_j} A_j^k X_j^{k*} = b$$

so $X = (X_1, X_2, \ldots, X_N)$ is a solution to (LP)

with value $\sum_{j=1}^{N} \sum_{k=1}^{k_j} C_j^k X_j^{k*}$.

QED

2.7 A Solution Procedure for the Two Duty Period Scheduling Problem

Proposition 2.1 has shown that (P*) in Section 2.5 is equivalent to (P). Therefore, a relaxation of (P) is

$$\begin{aligned} \text{(PR)} \quad \text{Min} \quad & C^1 X^1 + C^2 X^2 \\ \text{st} \quad & TA^1 X^1 + TA^2 X^2 = b \end{aligned}$$

which is simply a network flow problem. Since our original problem (P), the two duty period scheduling problem, is an integer programming problem; we can

incorporate the relaxation (PR) into a branch and bound implicit enumeration strategy. The following algorithm, in the notation of Geoffrion and Marsten (1972), outlines a solution procedure for the two duty period scheduling problem.

Algorithm 2.1

STEP 0 Initialize value of the best solution so far:

INCUMBENT = INFINITY

Initialize the level of the tree search:

LEVEL = 0

Let the initial candidate problem be the original problem:

(CP) = (P)

STEP 1 Solve (CPR), the relaxation of (CP).

If VALUE(CPR) $\geq$ INCUMBENT, GO TO STEP 2

If (CPR) solution is feasible in (P) then set INCUMBENT = VALUE(CPR) and GO TO STEP 2

LEVEL = LEVEL + 1

GO TO STEP 2

STEP 2 Create the next Candidate Problem (CP) at this level of the tree search.
If none exists, GO TO STEP 3
GO TO STEP 1

STEP 3 IF LEVEL = 0, STOP
Back up in tree:
Set LEVEL = LEVEL - 1
GO TO STEP 2

The relaxation (PR) can be used with Algorithm 2.1 for a solution procedure for the two duty period scheduling problem. For a significantly different approach to utilizing network relaxations for solving set partitioning problems see Mulvey (1975) and Glover and Mulvey (1976).

Mulvey (1975) reformulates set partitioning problems as transportation problems with side constraints (or equivalently as a generalized network problem). The number of arcs in the network flow problem is equal to the number of nonzero entries in the constraint matrix of the original set partitioning problem. Glover and Mulvey (1976) also reformulate the two duty period

scheduling problem as a network problem with side constraints. For each variable X_j with two segments they create two new nodes U_j and V_j and an "all-or-none" arc joining them. Rather than solve this network related version of the problem directly, the authors propose a heuristic solution technique.

The relaxation (PR) is very weak, to the point of being useless. However, we have great freedom in choosing $[C^1, C^2]$; the only stipulation being $C^1 + C^2 = C$. As we shall see in Chapter 4, by choosing $[C^1, C^2]$ properly, we can greatly strengthen the relaxation (PR). In fact with a moderate effort we can get (PR) to be almost as strong as the LP relaxation of (P).

PART II Implementation

CHAPTER 3

THE HELSINKI PROBLEM - THE RELAXATION

In this chapter we will discuss a special case of the two duty period scheduling problem - the two duty period set partitioning problem (or the Helsinki problem). The two segment characteristic of the problem allows us to use a minimum cost flow problem as a relaxation. The set partitioning characteristic means this relaxation is simply a shortest path problem. In addition, because the Helsinki problem is a set partitioning problem, we can reduce its dimensions by logical considerations. In Section 3.3 we present an algorithm for solving the two duty period set partitioning problem. In the next section we develop a branching procedure to be incorporated into this algorithm. In Sections 3.5 and 3.6 we introduce an algorithm for solving the shortest path relaxation.

3.1 The Helsinki Problem as a Shortest Path Problem with Coupled Arcs

A special case of the two duty period scheduling problem is the Helsinki Problem (HP)

$$
\begin{aligned}
\text{(HP)} \qquad & \text{Min } CX \\
& \text{st } AX = 1 \\
& \qquad X = 0,1
\end{aligned}
$$

where 1 is the N dimensional vector of ones and where each column of A consists of one or two segments of ones. For example, see Figure 3.1. It was in this form, a set partitioning problem, that Markku Tamminen of the Data Center of the Helsinki City Metropolitan Area, first brought the two duty period scheduling problem to the author's attention. The Helsinki problem arose in trying to schedule the city's bus crews using mathematical programming.

Applying to (HP) the transformations developed in Chapter 2 yields:

$$\text{Min} \quad \sum_{j=1}^{6} C_j X_j$$

st

X_1	X_2	X_3	X_4	X_5	X_6		RHS
1	0	1	1	0	0	=	1
0	0	1	1	0	0	=	1
0	1	0	0	1	0	=	1
0	1	0	0	1	1	=	1
0	1	1	0	0	1	=	1
1	0	1	1	0	1	=	1
1	0	0	1	1	0	=	1
0	1	0	0	1	0	=	1

$$X_j = 0,1 \qquad j = 1,2,\ldots,6$$

An Example of the Helsinki Problem

Figure 3.1

$$
\begin{aligned}
\text{(HP*)} \quad & \text{Min} && C^Y Y + C^Z Z \\
& \text{st} && (Y,Z) \text{ in } S \\
& && IY - IZ = 0
\end{aligned}
$$

where S is the set of solutions to a shortest path problem. For notational convenience we have replaced each variable X_j with Y_j (rather than X_j^1) and Z_j (rather than X_j^2). See Figure 3.2. Using the terminology of Geoffrion and Marsten (1972) we may write the relaxation of (HP*) as

$$
\begin{aligned}
\text{(HPR)} \quad & \text{Min} && C^Y Y + C^Z Z \\
& \text{st} && (Y,Z) \text{ in } S
\end{aligned}
$$

Then a feasible strategy for solving (HP) is to incorporate the relaxation (HPR) into a branch and bound procedure, such as Algorithm 2.1. The relaxation (HPR) is a shortest path problem with M + 1 nodes and (no more than) 2N arcs.

We employ the following terminology. We say that a column X_j in (HP) corresponds to two VARIABLES or ARCS Y_j and Z_j in (HP*) and (HPR). We refer to the coupled arcs Y_j and Z_j as PARTNERS. Two variables (of (HP*))

$$\text{Min} \quad \sum_{j=1}^{6} c_j^y Y_j + \sum_{j=1}^{5} c_j^z Z_j$$

st

Y_1	Z_1	Y_2	Z_2	Y_3	Z_3	Y_4	Z_4	Y_5	Z_5	Y_6		RHS
1	0	0	0	1	0	1	0	0	0	0	=	1
0	0	0	0	1	0	1	0	0	0	0	=	1
0	0	1	0	0	0	0	0	1	0	0	=	1
0	0	1	0	0	0	0	0	1	0	0	=	1
0	0	1	0	0	1	0	0	0	0	1	=	1
0	1	0	0	0	1	0	1	0	0	1	=	1
0	1	0	0	0	0	0	1	0	1	0	=	1
0	0	0	1	0	0	0	0	0	1	0	=	1

$$Y_j = Z_j \qquad j = 1,2,\ldots,5$$

$$Y_j = 0,1 \qquad j = 1,2,\ldots,6$$

$$Z_j = 0,1 \qquad j = 1,2,\ldots,5$$

where $c_j^y + c_j^z = c_j \quad j = 1,2,\ldots,6$

The Helsinki Problem of Figure 3.1 Reformulated with Decoupled Arcs

Figure 3.2

CONFLICT if there is a row covered by both variables (i.e. each column has a one in the same row). We use the terms RELAXATION and SHORTEST PATH PROBLEM interchangeably. Similarly the terms VARIABLE (of (HP*)) and ARC are used interchangeably, as are ROW and NODE (of the shortest path problem). Consequently, a variable (of (HP*)) covers a row if and only if the corresponding arc starts in or passes over that row. An arc starts at a node if and only if the corresponding segment of ones in the original Helsinki problem starts at the corresponding row. An arc ends at a node if and only if the corresponding segment of ones in the original Helsinki problem ends at the row preceding the corresponding row.

3.2 Logical Elimination for the Helsinki Problem

Algorithm 2.1 can be reformulated more specifically taking into account the special features of both the set partitioning problem (HP*) and the shortest path relaxation (HPR). Because the original problem (HP*) is a set partitioning problem, we know that once a variable is chosen (i.e. set equal to one), all variables which

conflict with it may be eliminated (i.e. set equal to zero).

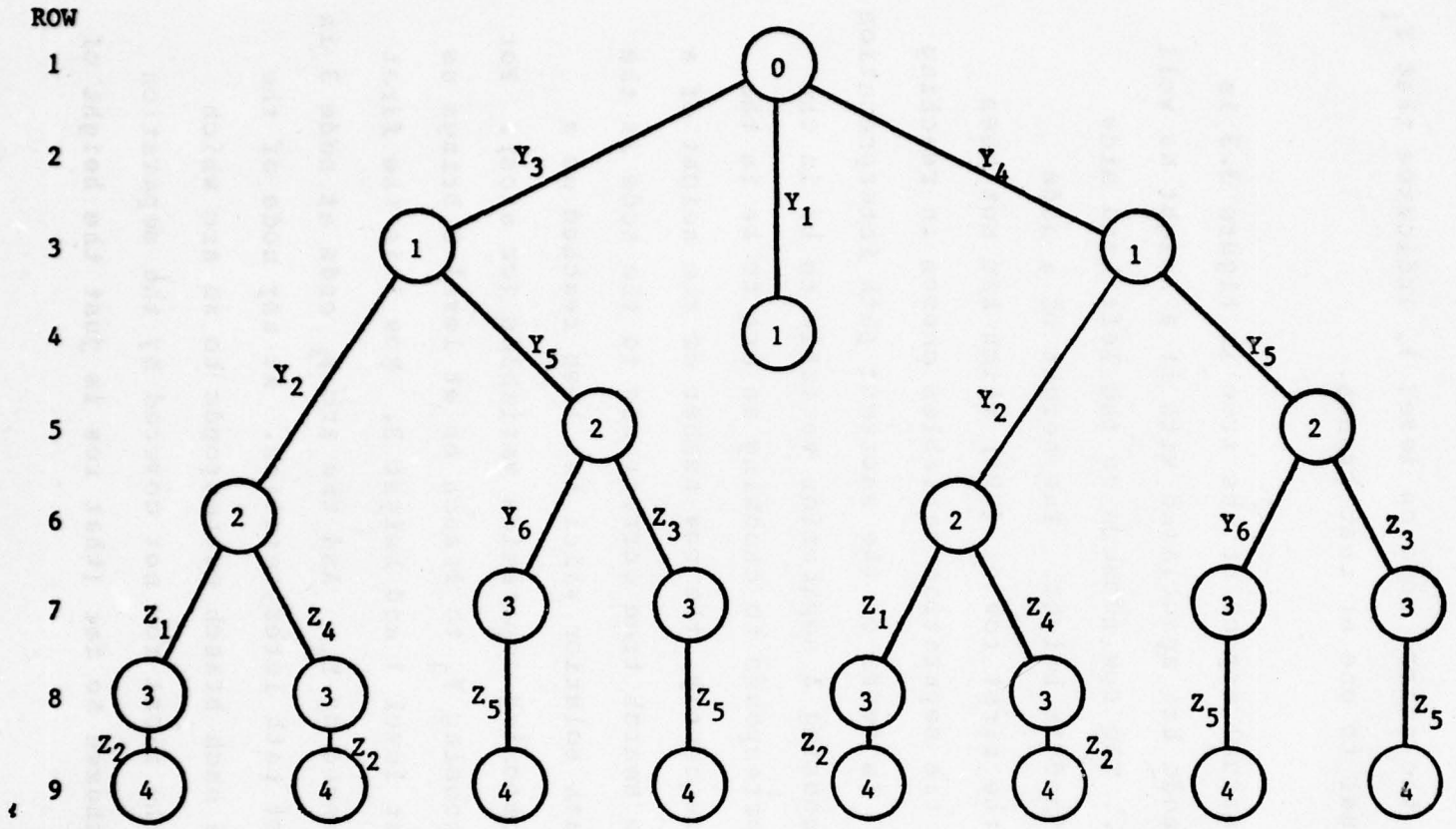
Similarly, we can take advantage of the fact that the relaxation is a shortest path problem and that it is obtained by decoupling arcs. Consequently, what is true of one arc is true of its partner. Therefore, when an arc is chosen, we can eliminate all the arcs which conflict with its partner as well as itself. Of course, when an arc is eliminated, its partner is also eliminated.

From the shortest path formulation it can be seen that if no arcs enter a node, then all arcs leaving that node may be eliminated. This is simply because no arc leaving that node can participate in a solution to the shortest path problem. Similarly, if no arcs leave a node, all arcs entering that node may be eliminated.

3.3 An Algorithm for Solving the Helsinki Problem

Algorithm 2.1 outlines the general procedure for a tree search. It is an interesting feature of the algorithm which we will now develop that a node in the search tree will often have more than two successor nodes. At any given node, rather than choosing a single separation variable (SEPVAR) to branch on, our algorithm will choose a number of variables in a manner we will describe in Section 3.4. They will correspond to the arcs starting at a given node in the shortest path relaxation. Each branch at a node will correspond to including exactly one of these arcs. See Figure 3.3 corresponding to the example in Figure 3.2.

We have employed some special notation in Figure 3.3. Each circle corresponds to a node of the search tree. The number within each circle represents the level on the search tree (i.e. the number of separation variables which have been chosen in reaching that node). Each arc in Figure 3.3 corresponds to a branch. The label on each arc indicates the separation variable at that branch. For example, the label Y_1, on the branch



Complete Search Tree for the Example in Figure 3.2

Figure 3.3

leading to the second node on level 1, indicates that Y_1 was set equal to one at that branch.

The unusual aspect of the tree in Figure 3.3 is that each node has associated with it a height as well as a level. The row numbers on the left hand side indicate a node's height. The height of a node indicates the first row of (HP*) which has not been covered by the separation variables chosen in reaching that node. In terms of the shortest path interpretation of (HP), choosing a separation variable to be in the solution corresponds to choosing an arc to be in the path. Consequently, the row number or the height of a node in the search tree corresponds to the node in the shortest path solution which has been reached as a result of choosing separation variables (or arcs). For example, choosing Y_3 to branch on at level 0 brings us to a node at level 1 and height 3. Row 3 is the first row not covered by Y_3. And the arc Y_3 ends at node 3 in the shortest path interpretation. At any node of the search tree each branch corresponds to an arc which begins in the first row not covered by the separation variables chosen so far (that row is just the height of the node).

Algorithm 3.1 reiterates, with greater specificity, Algorithm 2.1.

Algorithm 3.1

STEP 0 INCUMBENT = *INFINITY*

LEVEL = 0

(CP) = (HP)

(CPR) = (HPR)

STEP 1 Solve (CPR)

IF VALUE(CPR) $\geq$ INCUMBENT, GO TO STEP 2a

IF (CPR) solution satisfies Y = Z then

set INCUMBENT = VALUE(CPR) and

GO TO STEP 2a

LEVEL = LEVEL + 1

GO TO STEP 2b

STEP 2a Set SEPVAR(LEVEL) = PARTNER(SEPVAR(LEVEL)) = 0

Reinstate those variables eliminated at this LEVEL because they or their partners conflicted with SEPVAR(LEVEL) or its partner.

STEP 2b Create the next Candidate Problem (CP):

Choose a new separation variable for this level,

SEPVAR(LEVEL)

If none exists, GO TO STEP 3

Eliminate (i.e. set equal to zero) all

variables, and their partners, which

conflict with SEPVAR(LEVEL) and

PARTNER(SEPVAR(LEVEL))

Set SEPVAR(LEVEL) = PARTNER(SEPVAR(LEVEL)) = 1

GO TO STLP 1

STEP 3 IF LEVEL = 0, STOP

Back up in tree:

Reinstate all the separation variables

eliminated at this LEVEL

Set LEVEL = LEVEL - 1

GO TO STEP 2a

3.4 The Branching Procedure

The shortest path interpretation of the Helsinki problem suggests a natural branching procedure. The

original problem (HP) can be considered one of trying to construct a shortest path such that for each arc included in the path its partner is also included. Consider then, the branching procedure as one of trying to construct such a path. At the top level of the tree solve the relaxation. Then choose as the separation variable the arc in the relaxation solution which begins at node 1. See Figure 3.3.

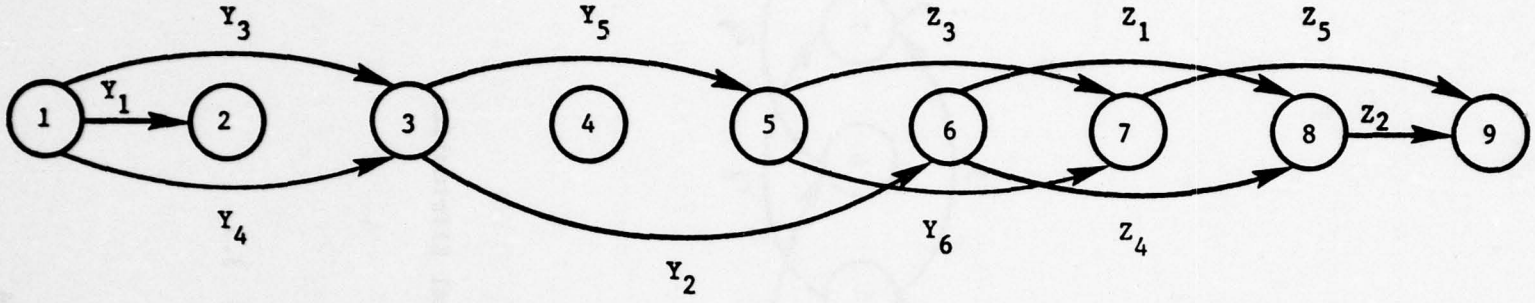
A similar process is used to choose a separation variable at all nodes of the decision tree. Each such node corresponds to a partial solution (the set of variables fixed to be in the solution by choosing separation variables at each level in reaching this node). To choose the next separation variable, solve the relaxation. Then choose the first arc, say Y_j, in the shortest path solution that has not already been chosen as a separation variable in reaching the present node of the decision tree. Then Y_j is the natural candidate for the separation variable. At the next level set Y_j (and Z_j) equal to one and repeat the process.

When a node is fathomed the separation variable at the present level is eliminated and the variables which were eliminated at this level, because they conflicted with this separation variable, are reinstated (i.e. they become free variables with no fixed value). The relaxation is resolved and a new separation variable is chosen, if the relaxation is feasible, in the manner described above.

This procedure is equivalent to binary branching (two branches from each node, one with the separation variable set equal to one, and the other equal to zero) in the sense that choosing the next separation variable at a level is equivalent to making a branch with the (previous) separation variable set equal to zero.

3.5 Solving the Relaxation

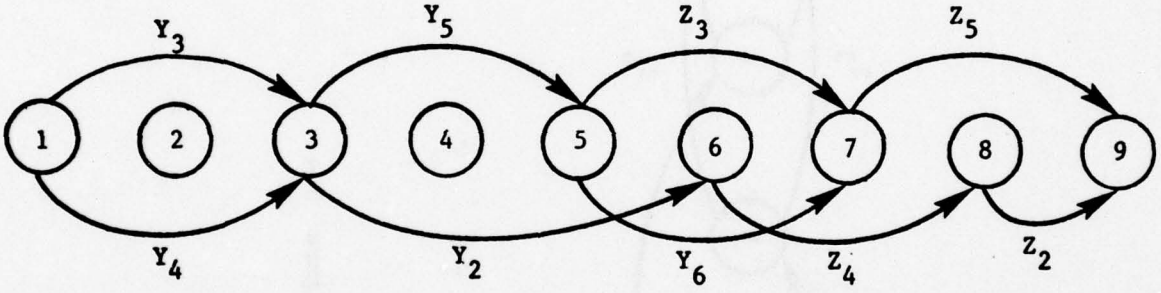
Step 2 of Algorithm 3.1 greatly reduces the size of the candidate problem. Consider the example of Figure 3.2. The corresponding shortest path problem is given in Figure 3.4. Since Node 2 has no exit, we may eliminate Y_1 and its partner Z_1 (see Figure 3.5). On



Graph of the Example given in Figure 3.2

Figure 3.4

Eliminate Y_1 and hence Z_1



Preliminary Logical Elimination

Figure 3.5

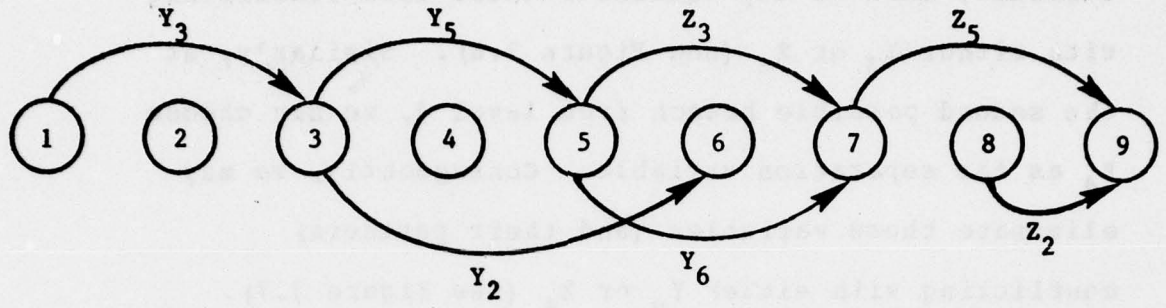
the first branch, if we choose Y_3 as the separation variable, then we may eliminate those arcs conflicting with either Y_3 or Z_3 (see Figure 3.6). Similarly, at the second possible branch from level 1, we may choose Y_4 as the separation variable. Consequently, we may eliminate those variables (and their partners) conflicting with either Y_4 or Z_4 (see Figure 3.7).

In the same manner, we can reduce the problem size by logical elimination at Step 1 of Algorithm 3.1, while solving the relaxation problem (CPR). Our relaxation is a shortest path problem which we will solve with a labeling algorithm. Consider the following forward reaching algorithm for solving a shortest path problem with M + 1 nodes [Denardo & Fox (1977)]:

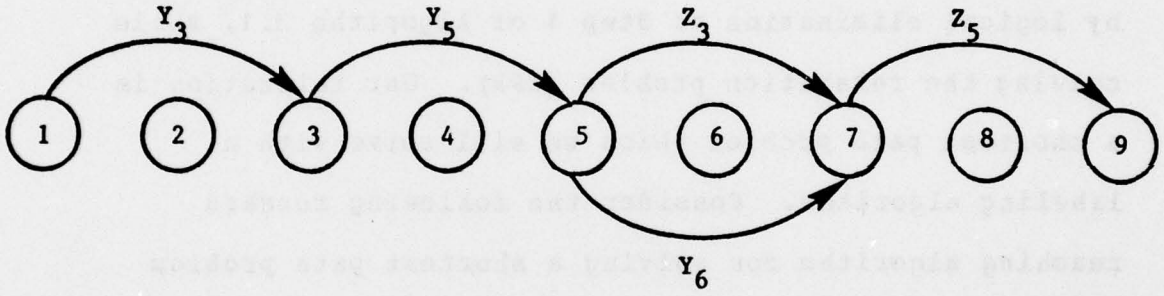
Algorithm 3.2

STEP 0 Let FORWARD LABEL(NODE) = INFINITY for each node except NODE = 1, FORWARD LABEL(1) = 0.
Let NODE = 1.

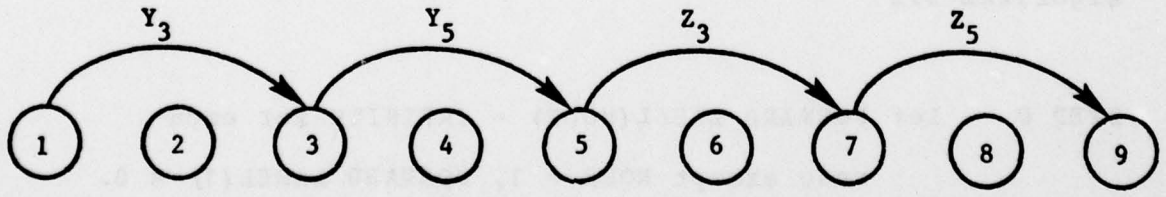
Choose Y_3 as separation variable and eliminate those arcs conflicting with Y_3: (Y_1), Y_4 and their partners: (Z_1), Z_4



Note: (6) has no exit, hence eliminate Y_2, Z_2



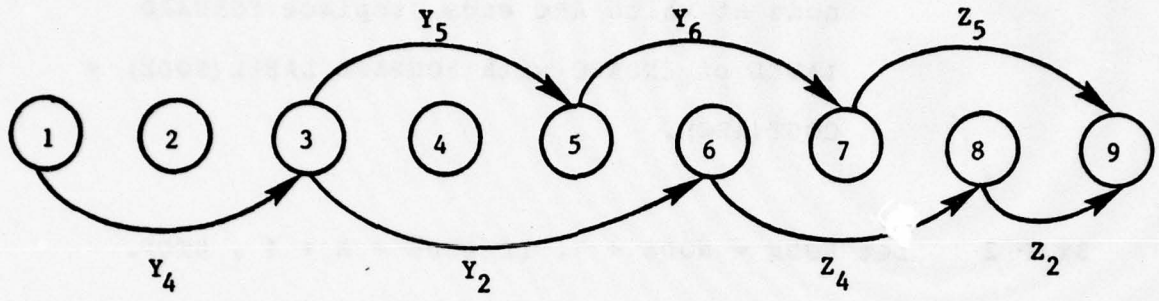
Y_3 chosen, so eliminate arcs conflicting with its partner Z_3: Y_6



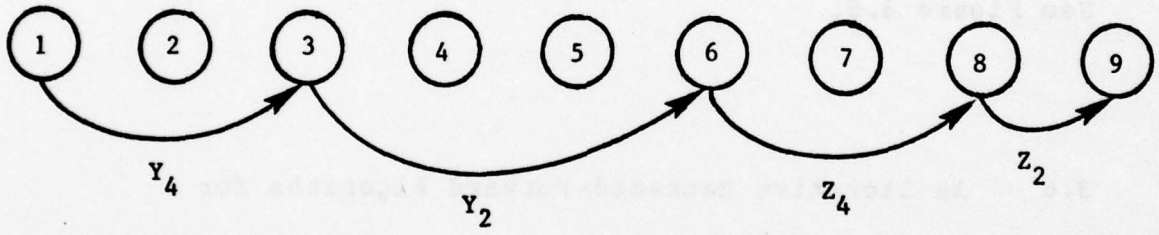
Branch 1 Level 0

Figure 3.6

Choose Y_4 as separation variable and eliminate those arcs conflicting with Y_4: (Y_1), Y_3 and their partners: (Z_1), Z_3



Y_4 chosen, so eliminate arcs conflicting with its partner Z_4: (Z_1), (Z_3), Z_5, Y_6 and their partners: (Y_1), (Y_3), Y_5



Branch 2 Level 0

Figure 3.7

STEP 1 For each arc which begins at NODE, if FORWARD LABEL(NODE) + COST(ARC) is less than the FORWARD LABEL for ENDARC, the node at which ARC ends, replace FORWARD LABEL of ENDARC with FORWARD LABEL(NODE) + COST(ARC).

STEP 2 Let NODE = NODE + 1. If NODE = M + 1 , STOP.

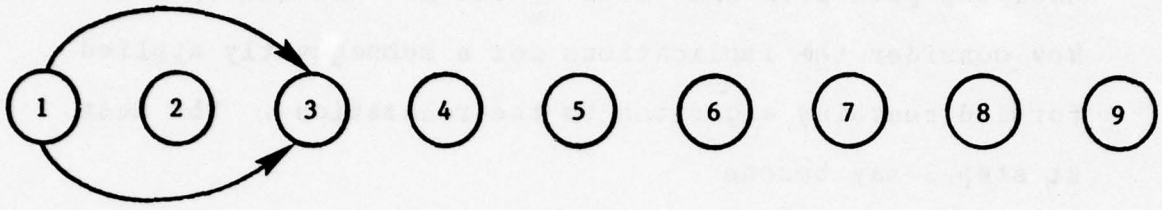
STEP 3 If FORWARD LABEL(NODE) < INFINITY, GO TO STEP 1. GO TO STEP 2

Similarly there is a backward reaching algorithm. See Figure 3.8.

3.6 An Iterative Backward-Forward Algorithm for Solving the Relaxation

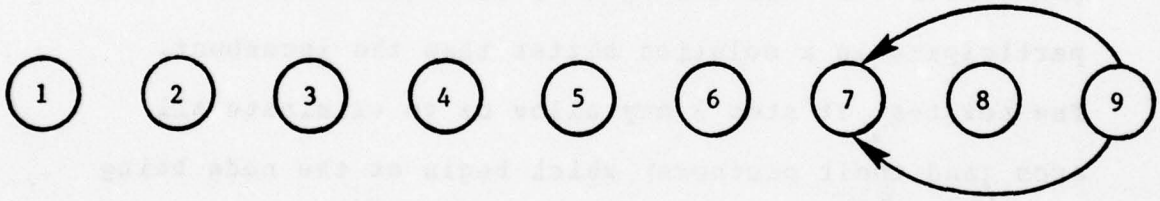
Note that if the branch and bound scheme has already found a feasible solution there is an INCUMBENT < INFINITY. In step 3, INFINITY may be replaced by INCUMBENT, making the algorithm stronger. However, an even stronger test may be applied at step 3.

Forward Reaching:



Generates a set of forward labels for each node j, j = 1,2,...,9, indicating the cost of the cheapest path from node 1 to node j.

Backward Reaching:



Generates a set of backward labels for each node j, j = 1,2,...9, indicating the cost of the cheapest path from node j to node 9.

Shortest Path Algorithms

Figure 3.8

Suppose the relaxation is solved first using the backward reaching algorithm. This would result in each node having a BACKWARD LABEL indicating the cost of the cheapest path from that node to the M + 1st (last) node. Now consider the implications for a subsequently applied forward reaching algorithm to the relaxation . The test at step 3 may become

If FORWARD LABEL(NODE) + BACKWARD LABEL(NODE) < INCUMBENT, GO TO STEP 1

This test checks each node, to see if it is possible for an arc ending or starting at that node to participate in a solution better than the incumbent. The new test at step 3 may allow us to eliminate all arcs (and their partners) which begin at the node being considered.

However, because of the ccupled arc nature of our shortest path relaxation, it is better to instead add a new test at step 1:

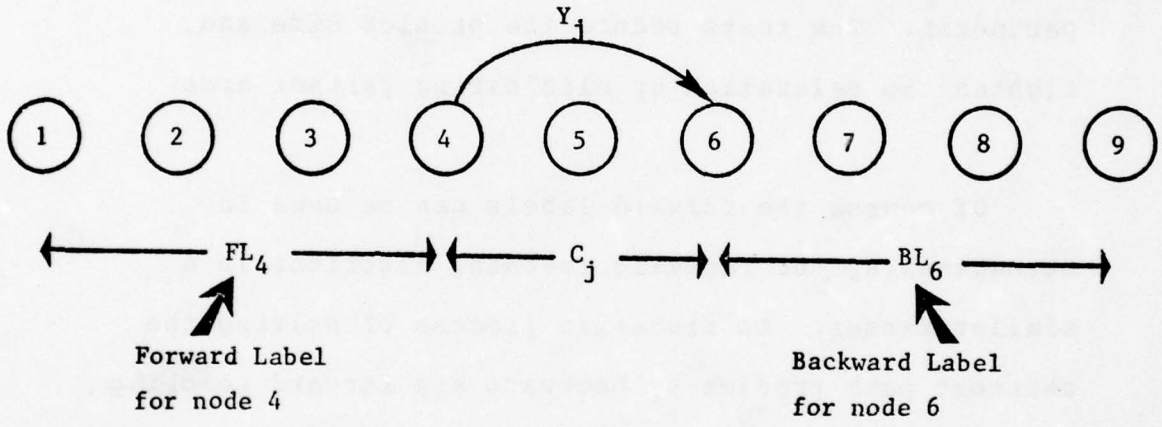
If FORWARD LABEL(NODE) + COST(ARC) + BACKWARD LABEL(ENDARC) $\geq$ INCUMBENT, eliminate ARC

This new test at step 1 may allow the ARC considered to be eliminated since it cannot participate in a solution of the (CPR) better than the INCUMBENT. When the arc is eliminated, its partner is also eliminated. This new test at step 1 makes the new test at step 3 redundant. See Figure 3.9.

The advantage of these new bounding tests is that they may eliminate arcs (and conseguently their partners). The tests reduce the problem size and tighten the relaxation by eliminating partner arcs.

Of course the forward labels can be used in strengthening the backward reaching algorithm in a similar manner. An iterative process of solving the shortest path problem by backward and forward reaching can be used until equilibrium is reached, (i.e. until no arcs are eliminated in a full cycle of the algorithm - one backward pass and one forward pass).

Algorithm 3.3 is the solution procedure for solving the shortest path relaxation (CPR) in Algorithm 3.1, which incorporates these ideas. FORWARD (BACKWARD) STOP is a flag to let us know whether or not the last forward



Arc Y_j may be eliminated if FORWARD LABEL(4) + COST(Y_j)
+ BACKWARD LABEL(6) is greater than INCUMBENT

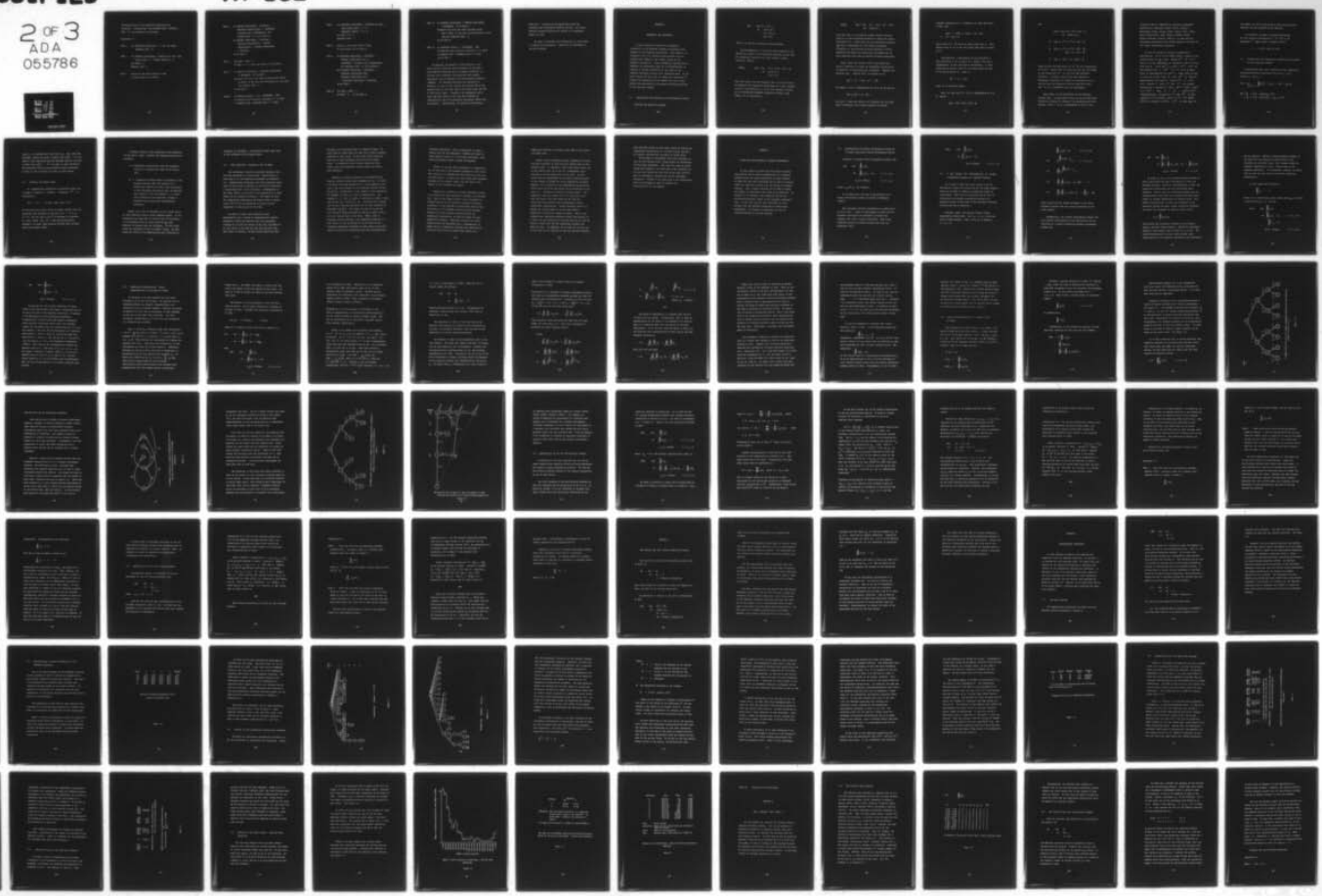
Iterative Backward-Forward Reaching

Figure 3.9

AD-A055 786 MASSACHUSETTS INST OF TECH CAMBRIDGE OPERATIONS RESE--ETC F/G 12/1
A LAGRANGEAN RELAXATION ALGORITHM FOR THE TWO DUTY PERIOD SCHED--ETC(U).
JUN 78 W B SHEPARDSON DAAG29-76-C-0064

UNCLASSIFIED TR-152 ARO-14261.9-M NL

2 OF 3
ADA
055786



(backward) pass of the algorithm eliminated any variables. Consequently, when FORWARD STOP = BACKWARD STOP = 1, the algorithm is finished.

Algorithm 3.3

STEP 0 Let BACKWARD LABEL(NODE) = 0 for all NODES
BACKWARD STOP = 0

STEP 1 Let FORWARD LABEL(NODE) = INFINITY for each node except NODE = 1, FORWARD LABEL(1) = 0.
Let NODE = 1.
FORWARD STOP = 1.

STEP 2 Choose an ARC which begins at NODE
If none exists go to STEP 4

STEP 3 If FORWARD LABEL(NODE) + COST(ARC) + BACKWARD LABEL(ENDARC) $\geq$ INCUMBENT, eliminate ARC, PARTNER(ARC), set FORWARD STOP = 0, GO TO STEP 2.

If FORWARD LABEL(NODE) + COST(ARC) < FORWARD LABEL(ENDARC), FORWARD LABEL(ENDARC) = FORWARD LABEL(NODE) + COST(ARC)

GO TO STEP 2.

STEP 4 Let NODE = NODE + 1

If NODE = M + 1 (the last node), GO TO STEP 6.

STEP 5 If FORWARD LABEL(NODE) + BACKWARD LABEL(NODE) < INCUMBENT, GO TO STEP 2

Eliminate all arcs and their partners which begin at NODE. If any arcs are eliminated at this step, set FORWARD STOP = 0

GO TO STEP 4

STEP 6 If FORWARD LABEL(M + 1) $\geq$ INCUMBENT, STOP

If shortest path solution satisfies Y = Z, STOP

If FORWARD STOP = BACKWARD STOP = 1, STOP.

STEP 7 Let BACKWARD LABEL(NODE) = INFINITY for each node except NODE = M + 1,
BACKWARD LABEL(M + 1) = 0.
Let NODE = M + 1
BACKWARD STOP = 1

STEP 8 Choose an arc which ends at NODE.
If none exists go to Step 10.

STEP 9 If BACKWARD LABEL(NODE) + COST(ARC) + FORWARD LABEL(BEGIN ARC) $\geq$ INCUMBENT, Eliminate ARC, PARTNER(ARC), set BACKWARD STOP = 0, GO TO STEP 8.
If BACKWARD LABEL(NODE) + COST(ARC) < BACKWARD LABEL(BEGIN ARC),
BACKWARD LABEL(BEGIN ARC)
= BACKWARD LABEL(NODE) + COST(ARC)
GO TO STEP 8.

STEP 10 Let NODE = NODE - 1
IF NODE = 1, GO TO STEP 12

STEP 11 If BACKWARD LABEL(NODE) + FORWARD LABEL(NODE) < INCUMBENT, GO TO STEP 8

Eliminate all arcs and their partners which end at NODE. If any arcs are eliminated at this step set BACKWARD STOP = 0.

GO TO STEP 10

STEP 12 If BACKWARD LABEL(1) $\geq$ INCUMBENT, STOP

If shortest path solution satisfies Y = Z, STOP.

If FORWARD STOP = BACKWARD STOP = 1, STOP.

GO TO STEP 1.

In applying the methods of this chapter it was found that the shortest path relaxation is too weak. This HELSINKI algortihm did not compare favorably, in solving large problems, with existing more general algorithms such as Marsten's Set Partitioning Algorithm (SETPAR). The HELSINKI algorithm was effective in reducing the size of the problem (by roughly 60% at the second level of the tree, 80% at the third level and 90% at the fourth level . However, the relaxation was so weak that most nodes in the tree were fathomed by infeasibility (due to eliminating variables) rather than by bounding. Consequently, the algorithm generated a

huge tree. In spite of the speed with which the shortest path subproblems could be solved; the entire Helsinki problem could not be solved in a reasonable amount of time.

In order to overcome this difficulty, we took steps to tighten the relaxation. These will be discussed in the next chapter.

CHAPTER 4

TIGHTENING THE RELAXATION

In this chapter we construct a Lagrangean relaxation of the Helsinki problem by dualizing with respect to the coupling constraints. This results in a relaxation that is a shortest path problem with arc lengths that depend on the current values of the Lagrange multipliers. Using subgradient optimization to maximize the Lagrangean leads to an interpretation of seeking an optimal allocation of the costs of the original variables between their decoupled arcs. In the final section we show that, by using this technique to tighten our relaxation we can reduce the size of the search tree generated by the branch and bound algorithm of the previous chapter.

4.1 Subgradient Optimization and the Helsinki Problem

Consider the Helsinki problem:

$$
\begin{aligned}
(HP) \qquad & \text{Min } C^Y Y + C^Z Z \\
& \text{st} \quad (Y,Z) \text{ in } S \\
& \qquad IY - IZ = 0 \\
& \qquad Y,Z = 0,1
\end{aligned}
$$

where S is the set of shortest path solutions.

By multiplying IY - IZ by the dual variables U and adding the result to the objective function, we can form the Lagrangean relaxation of (HP) [Everett (1963), Geoffrion (1974)]:

$$
\begin{aligned}
(HPR_U) \qquad W(U) = \text{Min} \quad & C^Y Y + C^Z Z + U(Y - Z) \\
\text{st} \quad & (Y,Z) \text{ in } S \\
& Y,Z = 0,1
\end{aligned}
$$

Note that (HPR_U) has the property that it is not altered by dropping the integrality conditions on Y and Z (indeed (Y,Z) in S guarantees Y,Z = 0,1). This feature is known as the INTEGRALITY PROPERTY [Geoffrion (1974)]. Then (HPR_U) can be rewritten as

$$(HPR_U) \qquad W(U) = \text{Min} \quad (C^Y + U)Y + (C^Z - U)Z$$
$$\text{st} \quad (Y,Z) \text{ in } S$$

Note that W(U) is a piecewise linear concave function since it is the pointwise minimum of a family of linear functions of U. Since W(U) has the integrality property $\max_U W(U)$ is equivalent to the linear programming relaxation of the Helsinki problem [Geoffrion (1974)]. Geoffrion has shown that Lagrangean relaxation can be very effective when used in branch and bound algorithms.

Held, Wolfe and Crowder (1974) have shown that using a technique now known as subgradient optimization is effective in maximizing the Lagrangean. Consider our function W(U). Suppose $(\hat{Y},\hat{Z})$ is optimal for $\hat{U}$:

$$W(\hat{U}) = (C^Y + \hat{U})\hat{Y} + (C^Z - \hat{U})\hat{Z}$$

We define s to be a SUBGRADIENT of W at $\hat{U}$ if and only if

$$W(U) \leq W(\hat{U}) + (U - \hat{U})s$$

for all U. Then the vector s at $\hat{U}$ points into the half space (orthogonal to s) which contains all better

(larger) solutions to W. Consider a U' such that W(U') > W($\hat{U}$), then

$$W(U') > W(\hat{U}) \geq W(U') - (U' - \hat{U})s$$

$$\Rightarrow \quad (U' - \hat{U})s > 0$$

which means (U' - $\hat{U}$) forms an acute angle with s. This implies that U' is in the half space into which s points at $\hat{U}$.

Consequently, a subgradient can be considered a good direction to go in search of a higher value of W. Consequently, we will maximize W iteratively. At each iteration we choose a new U^{t+1} by moving from our last U^t in the direction s^t. That is

$$U^{t+1} = U^t + a^t s^t$$

where a^t is the step length.

Now, we show that ($\hat{Y}$ - $\hat{Z}$) is a subgradient of W at $\hat{U}$. Recall

$$W(\hat{U}) = C^Y\hat{Y} + C^Z\hat{Z} + \hat{U}(\hat{Y} - \hat{Z})$$

Now

$$W(U) = \text{Min } C^Y Y + C^Z Z + U(Y - Z)$$
$$\text{st} \quad (Y,Z) \text{ in } S$$

$$\Rightarrow \quad W(U) \leq C^Y \hat{Y} + C^Z \hat{Z} + U(\hat{Y} - \hat{Z})$$

$$\Rightarrow \quad W(U) \leq C^Y \hat{Y} + C^Z \hat{Z} + \hat{U}(\hat{Y} - \hat{Z}) - \hat{U}(\hat{Y} - \hat{Z}) + U(\hat{Y} - \hat{Z})$$

$$\Rightarrow \quad W(U) \leq W(\hat{U}) + (U - \hat{U})(\hat{Y} - \hat{Z})$$

which is just the definition of $(\hat{Y} - \hat{Z})$ as a subgradient of W at $\hat{U}$. Notice that our proof of this did not depend on the constraints (Y - Z) put into the objective function. A general result holds that whenever a Lagrangean relaxation is formed, if constraints $AX \leq b$ are put into the objective function then the vector (AX - b) is a subgradient for the Lagrangean.

Take (HPR_U) as the relaxation of the Helsinki problem (HP). In particular (HPR_0) is the Shortest Path Relaxation obtained in chapter 2 by decoupling the arcs. However, since $\hat{Y} - \hat{Z}$ is a subgradient of W at $\hat{U}$, the

relaxation may be tightened by utilizing subgradient optimization techniques [Agmon (1954), Motzkin & Schoenberg (1954), Poljak (1967), Goffin (1971, 1976), Held & Karp (1971), Held, Wolfe & Crowder (1974), Fisher, Northup & Shapiro (1975)]. In this way the Lagrangean relaxation value should approach the value of the linear programming relaxation.

We use the following version of subgradient optimization. Let $\hat{W}$ be the TARGET VALUE, a guess at the optimal value of $\underset{U}{\text{Max}}\ W(U)$. Define $U^{t+1} \equiv U^t + a^t s^t$ where s^t is the subgradient at iteration t and a^t is the step length. Since $W(U^{t+1}) \leq W(U^t) + (U^{t+1} - U^t)s^t$ we may consider $W(U^t) + (U^{t+1} - U^t)s^t$ to be an upper bound or approximation for $W(U^{t+1})$. Then, since we want to reach our target value $\hat{W}$, pick U^{t+1} so that our approximation will equal $\hat{W}$. Choose $U^{t+1} = U^t + a^t s^t$ (i.e. choose a^t, since U^t and s^t are already determined) to satisfy $\hat{W} = W(U^t) + (U^{t+1} - U^t)s^t = W(U^t) + a^t |s^t|^2$. Then $a^t = [\hat{W} - W(U^t)]/|s^t|^2$. Computationally, solving for a^t and U^{t+1} at each iteration is very simple. Since $s^t = \hat{Y}^t - \hat{Z}^t$, (where $(\hat{Y}^t, \hat{Z}^t)$ is optimal in $W(U^t)$), $|s^t|^2$ is just equal to

the number of arcs in the shortest path solution whose partners are not included in the solution.

In practice, in order to assure convergence, we will include a multiplier d^t , $0 < d^t \leq 2$, in computing a^t [Held, Wolfe & Crowder (1974)]

$$a^t = d^t [\hat{W} - W(U^t)] / |s^t|^2$$

4.2 Interpreting the Subgradient Optimization Iteration for the Helsinki Problem

Conceptually then, each iteration of the subgradient optimization merely reallocates the cost C_j of X_j between Y_j and Z_j.

$$W(U) = \underset{(Y,Z) \text{ in } S}{\text{Min}} \sum_{j=1}^{N} [(C_j^Y + U_j)Y_j + (C_j^Z - U_j)Z_j]$$

$$U_j^{t+1} = U_j^t + d^t[\hat{W} - W(U^t)]s_j^t / |s^t|^2$$
$$= U_j^t + d^t[\hat{W} - W(U^t)](Y_j^t - \hat{Z}_j^t) / |s^t|^2$$

$$\text{So } (C_j^Y)^{t+1} = C_j^Y + U_j^{t+1}$$
$$= C_J^Y + U_j^t + d^t[\hat{W} - W(U^t)](\hat{Y}_j^t - \hat{Z}_j^t)/|s^t|^2$$
$$= (C_j^Y)^t + d^t[\hat{W} - W(U^t)](\hat{Y}_j^t - \hat{Z}_j^t)/|s^t|^2$$

$$\text{and } (C_j^Z)^{t+1} = (C_j^Z)^t - d^t[\hat{W} - W(U^t)](\hat{Y}_j^t - \hat{Z}_j^t)/|s^t|^2$$

If we let P^t be the penalty at iteration t + 1

$$P^t \equiv d^t[\hat{W} - W(U^t)]/|s^t|^2$$

$$\text{Then } (C_j^Y)^{t+1} = (C_j^Y)^t + P^t(\hat{Y}_j^t - \hat{Z}_j^t)$$

$$(C_j^Y)^{t+1} \begin{cases} = (C_j^Y)^t + P^t & \text{if } \hat{Y}_j^t = 1 \text{ and } \hat{Z}_j^t = 0 \\ = (C_j^Y)^t & \text{if } \hat{Y}_j^t = \hat{Z}_j^t \\ = (C_j^Y)^t - P^t & \text{if } \hat{Y}_j^t = 0 \text{ and } \hat{Z}_j^t = 1 \end{cases}$$

$$\text{And } (C_j^Z)^{t+1} \begin{cases} = (C_j^Z)^t - P^t & \text{if } \hat{Y}_j^t = 1 \text{ and } \hat{Z}_j^t = 0 \\ = (C_j^Z)^t & \text{if } \hat{Y}_j^t = \hat{Z}_j^t \\ = (C_j^Z)^t + P^t & \text{if } \hat{Y}_j^t = 0 \text{ and } \hat{Z}_j^t = 1 \end{cases}$$

For a variable X_j, if the partners Y_j and Z_j are both included in the shortest path relaxation solution or are both not included there is no adjustment in the allocation of C_j to C_j^Y and C_j^Z. However, if one partner

(say Y_j) is included and the other (Z_j) not, then the included partner is given a higher cost [$(C_j^Y)^t + P^t$] to make it less attractive and the excluded partner is given a lower cost [$(C_j^Z)^t - P^t$] to make it more attractive. The algorithm tries to make partners equally desirable so that if one is chosen the other is also chosen.

4.3 Choosing the Target Value

In implementing subgradient optimization there are a number of degrees of freedom. Generally U^{t+1} is determined by

$$U^{t+1} = U^t + d^t [(\hat{W} - W(U^t)] s^t / |s^t|^2$$

It has been shown [Held, Wolfe & Crowder (1974)] that the algorithm will converge to Max W(U) if $0 < d^t \leq 2$, $d^t \to 0$ and the sum of the d^t's diverges. In practice we have begun with $d^0 = 2$ and then periodically reduced it by half. This strategy follows that of Held, Wolfe and Crowder (1974).

A further choice in the algorithm is the selection of the target value. Solving the relaxation serves two purposes:

1. A relaxation value greater than the incumbent allows us to fathom that node of the search tree

2. A relaxation solution which is feasible in the original problem (HP) also allows us to fathom that node of the tree. This is because, in our Lagrangean relaxation W(U), we dualized with respect to the constraints Y = Z. Since these constraints are equations, finding a feasible answer guarantees complementary slackness.

Rather than take the incumbent as the target value we have generally taken a value somewhat higher. On the assumption that $\max_U W(U)$ is greater than the incumbent, it is reasonable that $\hat{W}$, which would be $\max_U W(U)$ ideally, be greater than the incumbent. We have tried using the incumbent as well as higher values. We have found the latter to be significantly more effective in

reaching the incumbent. In practice we have used 120% of the incumbent as the target value.

4.4 Tree Structure - Bounding a Set of Nodes

The Lagrangean relaxation approach developed here has been embedded in a tree search. Specifically we have used it as the relaxation procedure in Algorithm 3.3. Computational experience with this algorithm shows that for each level we descend in the tree we eliminate a large proportion of the variables. Consequently in the lower levels of the tree we are dealing with a small subset of the original variables. The result is that the subgradient optimization procedure tends to derive costs (C^Y, C^Z) for the uncoupled arcs that are not reasonable in other parts of the tree.

In order to combat this problem we have experimented with a number of implementation options. Etcheberry (1976) has reported that one successful strategy is to save the values of the dual variables U^t at each level of the tree and use them the next time that level is visited. We have tried saving the dual

variables and utilizing them in a number of ways. At each level we tried using the last set of dual variables obtained at that level. We have also tried using the last set of dual variables used at the next higher level. In addition we have tried a number of more exotic (and less effective) ways of using the stored dual variables.

However, we finally settled on a procedure which does not require saving dual variables at all. At level 0 we begin with dual variables $U^0 = 0$ and apportion the cost C_j of each variable in the original problem between C_j^Y and C_j^Z, proportional to the number of rows covered by the segment of ones in Y_j and Z_j respectively. For example if $C_j = 100$, Y_j covers 3 rows and Z_j covers 7 rows, then we set $C_j^Y = 30$ and $C_j^Z = 70$. From then on we simply use the most recently generated set of dual variables to determine $(C_j^Y, C_j^Z)^t$. The effect of this is to use the dual variable values obtained at the next higher level, as we are moving down in the tree. When a node is fathomed, before choosing a new separation variable at that level, we try to fathom all of the rest of the candidate separation variables at that level in one fell swoop by solving the relaxation with the old separation

variable eliminated. This is equivalent to doing a branch with the old separation variable set equal to zero, hence we refer to it as binary branching. This binary branching serves a number of purposes.

First, it is very often successful in eliminating all of the rest of the possible branches at that level. This allows us to immediately move back up in the tree one more level. If there are, say, eight more candidate branches at the current level, for the work of one branch we can eliminate all eight.

Second, this procedure of binary branching allows us to choose the next separation variable in a rational way. Even if the binary branch is not successful in eliminating all of the remaining eligible branching variables, it may succeed in eliminating some of them when there is an incumbent solution to the problem. Further, the best solution obtained during the subgradient optimization is used to choose the next separation variable. This best solution is a path composed from the decoupled arcs. Consequently, we simply use as a separation variable the first arc in this path which has not already been chosen as a

separation variable in reaching this node of the branch and bound tree.

Third, binary branching allows a mechanism to bring the dual variables in line with the optimal ones at the current level. In practice, when we are below level two in the search tree, we do not do a subgradient optimization as we are going down in the tree. We have found that the improvement in the dual variables gained is not enough to offset the extra computation involved. However, we do continue to use subgradient optimization below level two as we are moving up (or sideways) in the tree, i.e. whenever we perform a binary branch. Consequently, once past level two it is very easy to go very far down in the tree since we use only the relatively weak shortest path relaxations, without subgradient optimization. Binary branching gives us a method for getting back up in the tree without examining a prohibitive number of nodes. Rather than examine each remaining candidate branch individually at a level, we are often able to fathom them all at once by doing a branch with the old separation variable set equal to zero. In addition, as we work our way back up in the tree it is a method for getting the best possible

dual variable values at each level (since we always use subgradient optimization at a binary branch) and does not require storing dual variables at every level.

One problem we encountered with this technique is that we were dealing with a large number of subproblems of very small dimension. Over and over again we would reduce the problem to one with just a few variables but we could not fathom the node with either pure shortest path or subgradient optimization of the Lagrangean relaxation. Chapter 5 will deal with a new "relaxation" which we developed in order to handle this characteristic of the problem.

CHAPTER 5

A ROLE FOR PRIME NUMBERS IN INTEGER PROGRAMMING

In this chapter we show that any integer programming problem may be easily reformulated as a knapsack problem using the unique properties of prime numbers. We show that the same techniques may be used to linearly order the nodes of a search tree in solving an integer program. Such a linear ordering can be used to reduce the number of subproblems that need to be evaluated. In Section 5.5 we develop a new algorithm for the set partitioning problem, based on the knapsack representation. In the next section this algorithm is incorporated into the HELSINKI algorithm to fathom small subproblems. We present an algorithm for the set covering problem in the last section.

5.1 Reformulating an Integer Programming Problem as a Single Constraint Integer Programming Problem

Consider an integer linear programming problem (IP)

$$\text{(IP)} \qquad \text{Min} \quad \sum_{j=1}^{N} C_j X_j$$

$$\text{st} \quad \sum_{j=1}^{N} a_{ij} X_j = b_i \qquad i = 1,\ldots,M$$

$$X_j \geq 0, \text{ Integer} \qquad j = 1,\ldots,N$$

where a_{ij} and b_i are integers.

It is known that (IP) may be reformulated as a singly constrained problem [Garfinkel & Nemhauser (1972)].

The technique involves considering the constraints two at a time. A pair of multipliers is found and the weighted constraints are combined to replace the original pair with a single constraint. This transformation yields an integer program with only one constraint (IP').

$$(IP') \qquad \text{Min} \quad \sum_{j=1}^{N} C_j X_j$$

$$\text{st} \quad \sum_{j=1}^{N} \tilde{a}_j X_j = \tilde{b}$$

$$X_j \geq 0, \text{Integer} \qquad j = 1,\ldots,N$$

5.2 A New Method for Reformulating an Integer Programming Problem as a Knapsack Problem

It is easy to show that there exists a set of multipliers, namely the logarithms of the prime numbers, which may be used to combine an arbitrary number of constraints. This set of multipliers can be used to reformulate any integer programming problem as a knapsack problem (where some of the objective function coefficients may be negative).

Consider, again, the general integer linear programming problem (IP). Let P_i, $i = 1,\ldots,M$ be the first M prime numbers. Then, if X_j is an integer, $j = 1,\ldots,N$

$$\sum_{j=1}^{N} a_{ij}X_j = b_i \qquad i = 1,\ldots,M$$

$$\Leftrightarrow \qquad P_i^{\sum_{j=1}^{N} a_{ij}X_j - b_i} = 1 \qquad i = 1,\ldots,M$$

$$\Leftrightarrow \qquad \prod_{i=1}^{M} P_i^{\sum_{j=1}^{N} a_{ij}X_j - b_i} = 1$$

$$\Leftrightarrow \qquad \sum_{i=1}^{M} \left(\sum_{j=1}^{N} a_{ij}X_j - b_i\right) \ln P_i = 0$$

$$\Leftrightarrow \qquad \sum_{j=1}^{N} \left(\sum_{i=1}^{M} a_{ij} \ln P_i\right) X_j = \sum_{i=1}^{M} b_i \ln P_i$$

where going from the second statement to the third statement depends upon the unique properties of the prime numbers.

Consequently, any integer programming problem (IP) with integer coefficients in the constraints, can be rewritten as a singly constrained integer programming Problem (K).

$$(K) \qquad \text{Min} \quad \sum_{j=1}^{N} C_j X_j$$

$$\text{st} \quad \sum_{j=1}^{N} \left(\sum_{i=1}^{M} a_{ij} \ln P_i \right) X_j = \sum_{i=1}^{M} b_i \ln P_i$$

$$X_j \geq 0, \text{ Integer} \qquad j = 1,\ldots,N$$

We refer to (K) as the knifedge problem because of three characteristics which distinguish it from the knapsack problem. The first characteristic is that the constraint coefficients are not the desired small integers. Rather, they are irrational numbers and any transformation which does not sacrifice accuracy would result in integer coefficients of infinite size. The second characteristic is that the constraint is an equation, making the usual linear relaxation (obtained by taking the variables in order of their ratios

$$C_j / \sum_{i=1}^{M} a_{ij} \ln P_i$$

and setting the fractional variable to the largest smaller integer) almost useless. Indeed the constraint defines a fine "edge" with no width to it at all. The third characteristic is that, quite likely, some coefficients of the objective function or the constraint

may be negative. Unlike a normal knapsack problem, if the objective and constraint coefficients of any variable are of unlike signs, that variable cannot be assigned immediately a value. This is due to the equation constraint. It is possible, however, to modify (IP) so that (K) will have all positive constraint coefficients.

To (IP) append the constraint

$$\sum_{j=1}^{N+1} X_j = B$$

where B is a sufficiently large number and X_{N+1} is a slack variable with $C_{N+1} = 0$. We have

$$\text{(IP')} \qquad \text{Min} \sum_{j=1}^{N+1} C_j X_j$$

$$\text{st} \sum_{j=1}^{N+1} a_{ij} X_j = b_i \qquad i = 1,\ldots,M$$

$$\sum_{j=1}^{N+1} X_j = B$$

$$X_j \geq 0, \text{ Integer} \qquad j = 1,\ldots,N$$

Let P_{M+1} be a sufficienly large prime number such that

$$P_{M+1} \neq P_i \qquad i = 1,\ldots,M$$

and

$$\sum_{i=1}^{M} a_{ij} \ln P_i + \ln P_{M+1} > 0 \qquad j = 1,\ldots,N$$

using (IP'), the resulting knifedge problem (K') will have nonnegative constraint coefficients. In practice it will usually be more productive to achieve the same result in another manner. Suppose the constraint coefficient of X_j is negative. Then, if U_j is an upper bound for X_j (in practice an upper bound can usually be assumed), X_j may be replaced by $U_j - X_j$. This, too, will result in a knifedge formulation with nonnegative constraint coefficients.

Consider now (K), the Knifedge interpretation of a general integer programming problem. For ease of notation we write (K) as

$$(K) \qquad \text{Min} \quad \sum_{j=1}^{N} c_j X_j$$

$$\text{st} \quad \sum_{j=1}^{N} \tilde{a}_j X_j = \tilde{b}$$

$$X_j \geq 0, \text{ Integer} \qquad j = 1,\ldots,N$$

In general (K) can be quite difficult to solve. Since linear programming (and other) relaxations tend to be very weak due to the equality constraint, it appears to be desirable to avoid relaxation procedures. Alternatively, shortest path and dynamic programming offer solution techniques which avoid this problem. Assume for the moment that the coefficients $\tilde{a}_j$, $\tilde{b}$ of (K) are integer. Then (K) may be interpreted as a shortest path problem. Let G = (V,A) be a directed graph with vertex set $V = \{0,1,2,\ldots,\tilde{b}\}$ and arc set $A = \{(i,k): k-i = \tilde{a}_j \text{ for some } j,\ j = 1,\ldots,N\}$ [Garfinkel & Nemhauser (1972)]. Then solving (K) corresponds to finding the shortest path in G. Note that if $\tilde{a}_j > 0$, $j = 1,\ldots,N$, the graph G contains no cycles, since $\tilde{a}_j > 0$ for all j implies that if (i,k) is in A, then $k > i$. We will always assume $\tilde{a}_j > 0$ in (K) without loss of generality. Then (K) may be represented as an acyclic shortest path problem.

5.3 Numerical Considerations - Finite Approximations to the Logs of Primes

In section 5.2 we have assumed that the coefficients $\tilde{a}_j, \tilde{b}$ of (K) are integer. In practice this is reasonable since any computer implementation will require the use of rational numbers. However, our proof in Section 5.2 that (K) is equivalent to (IP) depended on the use of the exact logs of primes. (K) with irrational coefficients $\tilde{a}_j$, $\tilde{b}$ may still be represented as a shortest path problem.

Let G = (V,A) be a directed graph with vertex set V = $\{v: v = \sum_{j \in S} \tilde{a}_j X_j$ and $v \leq \tilde{b}$ where $S \subseteq \{1,2,\ldots,N\}$, $X_j \geq 0$, integer$\}$ and arc set A = $\{(i,k): k - i = \tilde{a}_j$ for some j = $1,2,\ldots,N\}$. Then solving (K) corresponds to finding the shortest path in G. Note that each variable X_j in (IP) corresponds to an arc X_j^v at each vertex v. Therefore, choosing an arc X_j^v to be in the path is equivalent to incrementing by one the value of the variable X_j. We define the SPAN of an arc X_j to be $\tilde{a}_j = \sum_{i=1}^{M} a_{ij} \ln P_i$. (The natural concept here is one of length - unfortunately early practitioners with shortest path problems gave the word length another connotation,

namely cost). We define the span of a path to be the sum of the spans of the arcs making up that path. The span of a node is simply the span of any path reaching that node.

In practice it is not possible to use the exact logs of primes. Are we then, justified in solving (K) in place of (IP)? Consider the following reformulation of (IP).

$$\text{Let } P_i^* = [\,10^t \ln P_i\,] \qquad t \text{ integer}$$

where [a] = largest integer less than or equal to a.

$$\text{Let} \quad a_j^* = \sum_{i=1}^{M} a_{ij} P_i^* + P_{M+1}^*$$

$$b^* = \sum_{i=1}^{M} b_i P_i^* + BP_{M+1}^*$$

Then

$$(K^*) \qquad \text{Min} \quad \sum_{j=1}^{N} C_j X_j$$

$$\text{st} \quad \sum_{j=1}^{N} a_j^* X_j = b^*$$

$$X_j \geq 0, \text{ Integer} \qquad j = 1,\ldots,N$$

is a relaxation of (IP). Moreover it is a relaxation which can be made arbitrarily close to (K) in the natural sense by choosing t large. (Another way of tightening the relaxation (K*) would be to use surrogate duality [Glover (1968, 1975), Greenberg & Pierskalla (1970), Karwan & Rardin (1976)].)

Proposition 5.1: If we assume X is bounded in (K), then by approximating the irrational coefficients $\tilde{a}_j, \tilde{b}$ of (K) closely enough by rational numbers a^*_j, b^*; (K) may be reformulated as an equivalent knapsack problem with rational coefficients.

Proof: If we assume the variables X are bounded, $X \in S = \{X: 0 \leq X_j \leq U_j,\ X_j \text{ Integer},\ j = 1,\ldots,N\}$, then S is finite. Let $S' = \{X \in S: \tilde{a}X \neq \tilde{b}\}$. For any $X \in S'$, $(\tilde{a}X - \tilde{b}) \neq 0$, so there exists a $\delta > 0$ and a neighborhood $N_X(\tilde{a})$ such that the absolute value of $(aX - \tilde{b})$ is greater than $\delta > 0$ for all $a \in N_X(\tilde{a})$. Then there exists a neighborhood $N_X(\tilde{b})$ such that $aX \neq b$ for all a in $N_X(\tilde{a})$ and all b in $N_X(\tilde{b})$. Let $N(\tilde{a}) = \bigcap_{X \in S'} N_X(\tilde{a})$ and $N(\tilde{b}) = \bigcap_{X \in S'} N_X(\tilde{b})$. Then $aX \neq b$ for all $a \in N(\tilde{a})$ and $b \in N(\tilde{b})$ and $X \in S'$. Consider a set of rational coefficients $\{(a^k, b^k) \in R^{N+1}: a^k_j, b^k \text{ rational},\ j = 1,2,\ldots,N;$

$k = 1,2,\ldots\}$ converging to $(\tilde{a},\tilde{b})$. Then all but a finite number of problems

$$(P^k) \quad \text{Min} \quad CX$$

$$\text{st} \quad \sum_{j=1}^{N} a_j^k X_j = b^k$$

define problems equivalent to (K). Consequently by choosing t large enough (but finite), (K*) will be equivalent to (K).

The advantage of (K*) is that the size of the shortest path problem (b* nodes) may be controlled by the user. By properly choosing t and the base of the logarithms, b* can be chosen arbitrarily.

By choosing t large we may approximate (IP) or (K) very closely. How does (K*) differ from (K)? To answer this we must develop a bit of machinery. Consider now the shortest path interpretation of (K), the knifedge representation of (IP). Choosing an arc X_j^v to be in the path corresponds to using a certain amount of resources in the original problem (IP) represented by the vector A_j. We shall prove in Proposition 5.2, that a path of

span P corresponds to a unique vector of resource utilization in (IP).

Proposition 5.2: Consider an integer programming problem (IP) and the corresponding knifedge problem (K) with its shortest path interpretation. Consider two sets of arcs $S_1 = \{X_{j_k}^{v_k}: k = 1,2,\ldots,k_1\}$ and $S_2 = \{X_{h_k}^{u_k}: k = 1,\ldots,k_2\}$
If the two sets of arcs have equal span,

$$\text{i.e. } \sum_{k=1}^{k_1} \sum_{i=1}^{M} a_{ij_k} \ln P_i = \sum_{k=1}^{k_2} \sum_{i=1}^{M} a_{ih_k} \ln P_i$$

then they must cover precisely the same rows the same number of times each, i.e. they must correspond to precisely the same resource vector.

Proof:

$$\sum_{k=1}^{k_1} \sum_{i=1}^{M} a_{ij_k} \ln P_i = \sum_{k=1}^{k_2} \sum_{i=1}^{M} a_{ih_k} \ln P_i$$

$$\Rightarrow \prod_{k=1}^{k_1} \prod_{i=1}^{M} P_i^{a_{ij_k}} = \prod_{k=1}^{k_2} \prod_{i=1}^{M} P_i^{a_{ih_k}}$$

$$\Rightarrow \prod_{i=1}^{M} P_i^{\sum_{k=1}^{k_1} a_{ij_k}} = \prod_{i=1}^{M} P_i^{\sum_{k=1}^{k_2} a_{ih_k}}$$

$$\Rightarrow \quad P_i^{\sum_{k=1}^{k_1} a_{ij_k}} = P_i^{\sum_{k=1}^{k_2} a_{ih_k}} \qquad i = 1,\ldots,M$$

$$\Rightarrow \quad \sum_{k=1}^{k_1} a_{ij_k} = \sum_{k=1}^{k_2} a_{ih_k} \qquad i = 1,\ldots,M$$

(since a_{ij} Integer)

QED

The proof of Proposition 5.2 depends upon the use of exact logs of primes. Consequently, (K*) is only an approximation of (K) since it is possible for a path of span P to represent more than one vector of resource utilization. It is the case that two paths S_1 and S_2 in the shortest path representation of (K*) having the same resource utilization

$$\text{i.e.} \quad \sum_{X_j^v \epsilon S_1} \sum_{i=1}^{M} a_{ij} = \sum_{X_j^v \epsilon S_2} \sum_{i=1}^{M} a_{ij}$$

will have the same span

$$\text{i.e.} \quad \sum_{X_j^v \epsilon S_1} \sum_{i=1}^{M} a_{ij} \; P_i^* = \sum_{X_j^v \epsilon S_2} \sum_{i=1}^{M} a_{ij} \; P_i^*$$

Where (K*) fails is that it may find an optimal solution S that is not feasible in (IP). That is, the utilization vector of the path corresponding to S may not be equal to b, the right hand side vector in the constraints of the original integer programming problem. This is because, due to approximating the logs of primes, two paths of equal span may not correspond to the same utilization of resources vector. We regard (K*) as having a resolution problem. With t very small (say 3) the nodes of the shortest path representation tend to become blurred together and the algorithm may perceive two actually distinct nodes as being one and the same node. Increasing t increases the algorithm's power of resolution.

This suggests a method for solving the relaxation (K*) in a manner that ensures it will be an equivalent problem to (IP). Suppose (K*) is being solved as a shortest path problem by a forward reaching algorithm (e.g. Algorithm 3.2). At a node (with span) j we check each arc originating at j. Let the first arc be i. Then arc i plus the path to j form a path to (j + span of arc i), (say k). If node k has a label less than infinity, we now require that the algorithm check that

the incumbent path to k and the new path (to j then k via i) have the same resource utilization vector. If they do, we proceed as usual. If they do *not*, then we arbitrarily give the new path (to j then k via i) span k + 1 and begin again with arc i. Provided t is reasonably large, this method should resolve all but the most pathological cases. In practice we have used t = 11 or 13 and have not used the above refinement to our algorithm as its computational burden is very high.

It is also possible to interpret (K*) from a geometric point of view. In the knifedge problem (K) the constraint

$$\sum_{j=1}^{N} \tilde{a}_j X_j = \tilde{b}$$

represents a hyperplane H in R^N. In all of R^N the only integer points in H are those that are feasible in (IP). And H includes all (IP) feasible integer points. The constraint

$$\sum_{j=1}^{N} a^*_j X_j = b^*$$

in (K*) may be seen as a hyperplane H* approximating H. H* is an approximation in that it still contains all (IP) feasible integer points but may contain additional integer points as well. Consequently, it is a tilted

version of H, where H* has been rotated slightly about the axis which is the affine subset of R^N defined by the constraints of (IP). Since H is a finite distance from all (IP) infeasible integer points it may be rotated a finite amount about this axis without including any non-feasible integer points. This corresponds exactly to choosing t large enough that H* rotates little enough to include no non-feasible integer points.

5.4 Induced Linear Orderings in Combinatorial Problems

The transformation which took us from (IP) to (K) may be looked at as one that induces a linear ordering on the set of partial solutions $\{\bar{X}: \bar{X} = \{\bar{X}_1, \bar{X}_2, \ldots, \bar{X}_k\}, k \leq N\}$. More precisely, it may be seen as inducing a linear one-to-one mapping from the vectors A_j in Z^M in resource space to the real numbers.

$$P: Z^M \rightarrow R$$

$$P(z) = \sum_{i=1}^{M} z_i \ln P_i$$

$$P(A_j) = \sum_{i=1}^{M} a_{ij} \ln P_i$$

Consider a partial solution $\bar{X}$ to (IP), $\bar{X} = \{\bar{X}_1, \bar{X}_2, \ldots, \bar{X}_k\}$, where for ease of notation the variables X_j have been reordered so that the variables with assigned values (i.e. in the partial solution) are $j = 1, 2, \ldots, k$. Then $\bar{X}$ uses a certain amount of resources, namely

$$\sum_{j=1}^{k} a_{ij} \bar{X}_j \qquad\qquad i = 1, \ldots, M$$

or equivalently,

$$\sum_{j=1}^{k} A_j \bar{X}_j$$

Consequently, we can extend the mapping P to map (partial) solutions of (IP) into the real numbers.

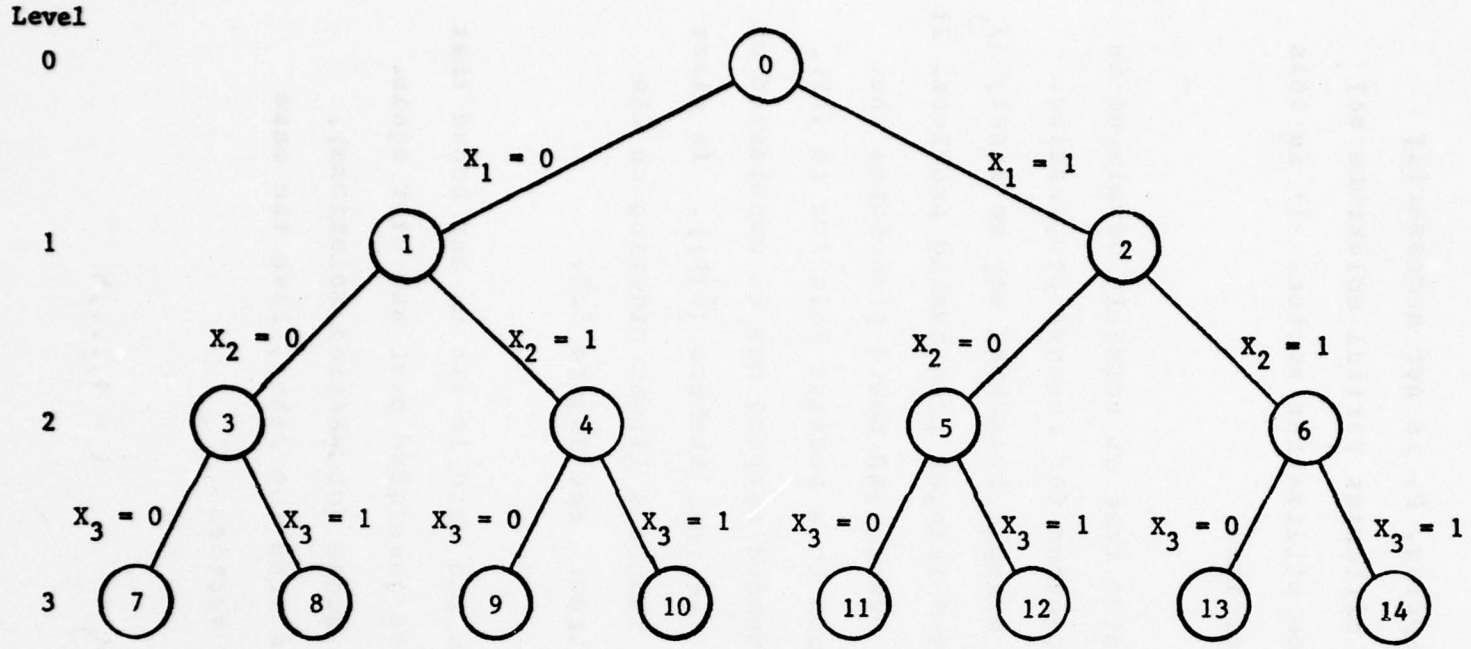
$$\begin{aligned} \bar{P}(\bar{X}) &= P\left(\sum_{j=1}^{k} A_j \bar{X}_j\right) \\ &= \sum_{j=1}^{k} P(A_j) \bar{X}_j \\ &= \sum_{j=1}^{k} \left(\sum_{i=1}^{M} a_{ij} \ln P_i\right) \bar{X}_j \end{aligned}$$

This extended mapping, $\bar{P}$, is not necessarily one-to-one since two different partial solutions may have the same resource utilization vector. It is this fact which may be exploited.

Consider a decision tree as normally developed in branch and bound procedures for integer programming. See Figure 5.1. For example branching may be binary ($X_j = 0$ and $X_j = 1$) for 0-1 integer programming problems. It is characteristic of branch and bound procedures that each node k corresponds to a partial solution to (IP). Consequently, our extended mapping can be considered as mapping each node to the real numbers ($\bar{P}(k)$). In other words we are able to induce a linear ordering on the nodes of the search tree. See Figure 5.2.

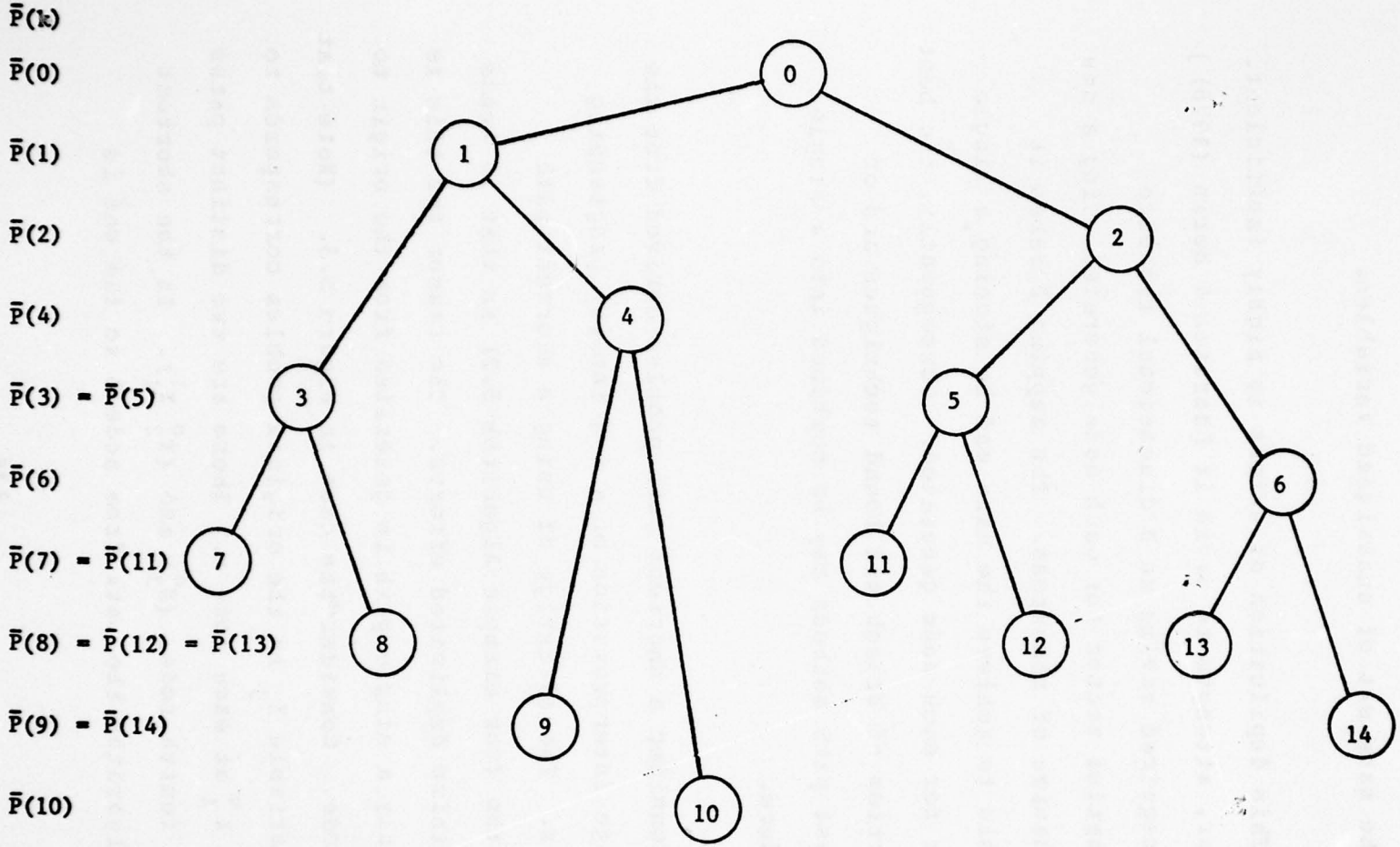
It is very often the case in branch and bound that identical subtrees are generated over and over again. This occurs when two nodes (or partial solutions), neither one descended from the other, have the same resource utilization vectors

i.e. $$\sum_{\bar{X}_j \epsilon \bar{X}} a_{ij} \bar{X}_j \qquad i = 1,\ldots,M$$



A Binary Search Tree for a 0-1 Integer Programming Problem where the Separation Variable at Level j is X_j

Figure 5.1



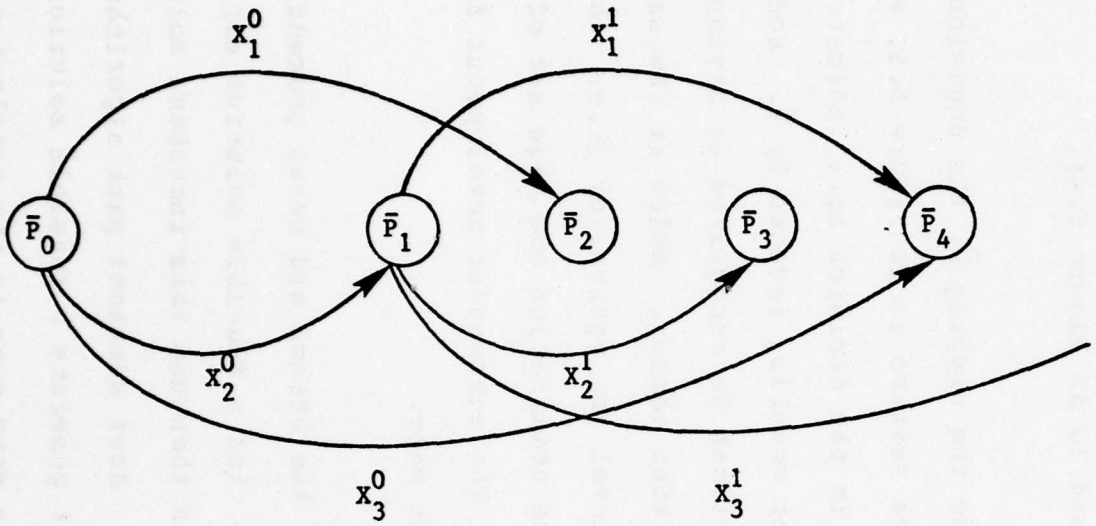
The Binary Search Tree of Figure 5.1 where the Height of a Node in the Tree Corresponds to its Position in the Linear Ordering Induced by $\bar{P}$

Figure 5.2

and the same set of unassigned variables.

This duplication of effort is highly inefficient. However, attempts to avoid it [Marsten & Morin (1978)] have required saving an M dimensional resource utilization vector for each node generated using a new combination of resources. The mapping $\bar{P}$ makes it possible to achieve the same end by storing a single number for each node generated. Consequently, the best properties of branch and bound techniques and of shortest path methods may be combined into a single procedure.

Consider a shortest path problem derived from the knifedge interpretation of a 0-1 integer programming problem. The advantage of using a shortest path algorithm (for example Algorithm 3.2) is that it tends to minimize duplicated efforts. The reason for this is that only a single path is generated from the origin to each node. Consider the case in Figure 5.3. (Note that each variable X_j in the original problem corresponds to an arc X_j^v at each node v.) There are two distinct paths to the fourth node, (X_3^o) and $(X_2^o\ X_1^1)$. In the shortest path algorithm the path from node 4 to the end is



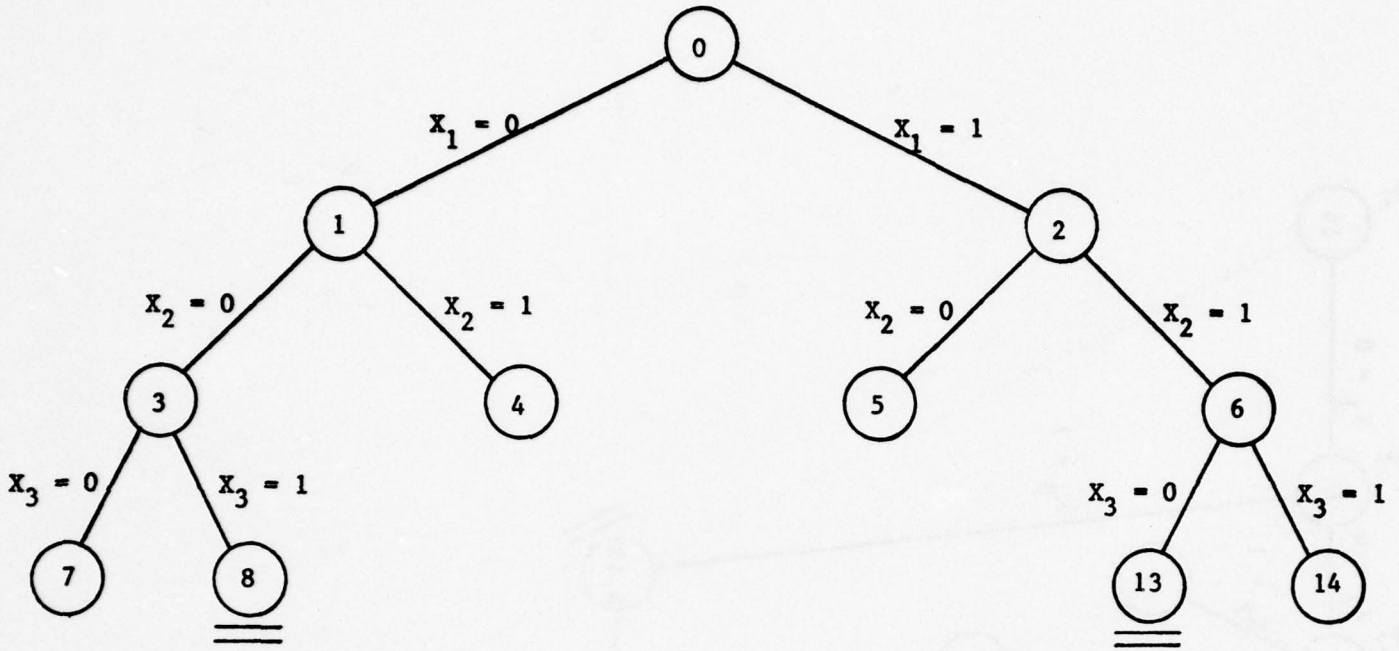
An Example of Solving a 0-1 Integer Programming Problem with a Shortest Path Interpretation of the Knifedge Formulation

Figure 5.3

considered only once. But in a binary search tree where X_j is the separation variable at level j (see Figure 5.4), the path from node 4 (of the shortest path representation) to the end would have to be considered twice (from nodes 8 and 13 in Figure 5.4).

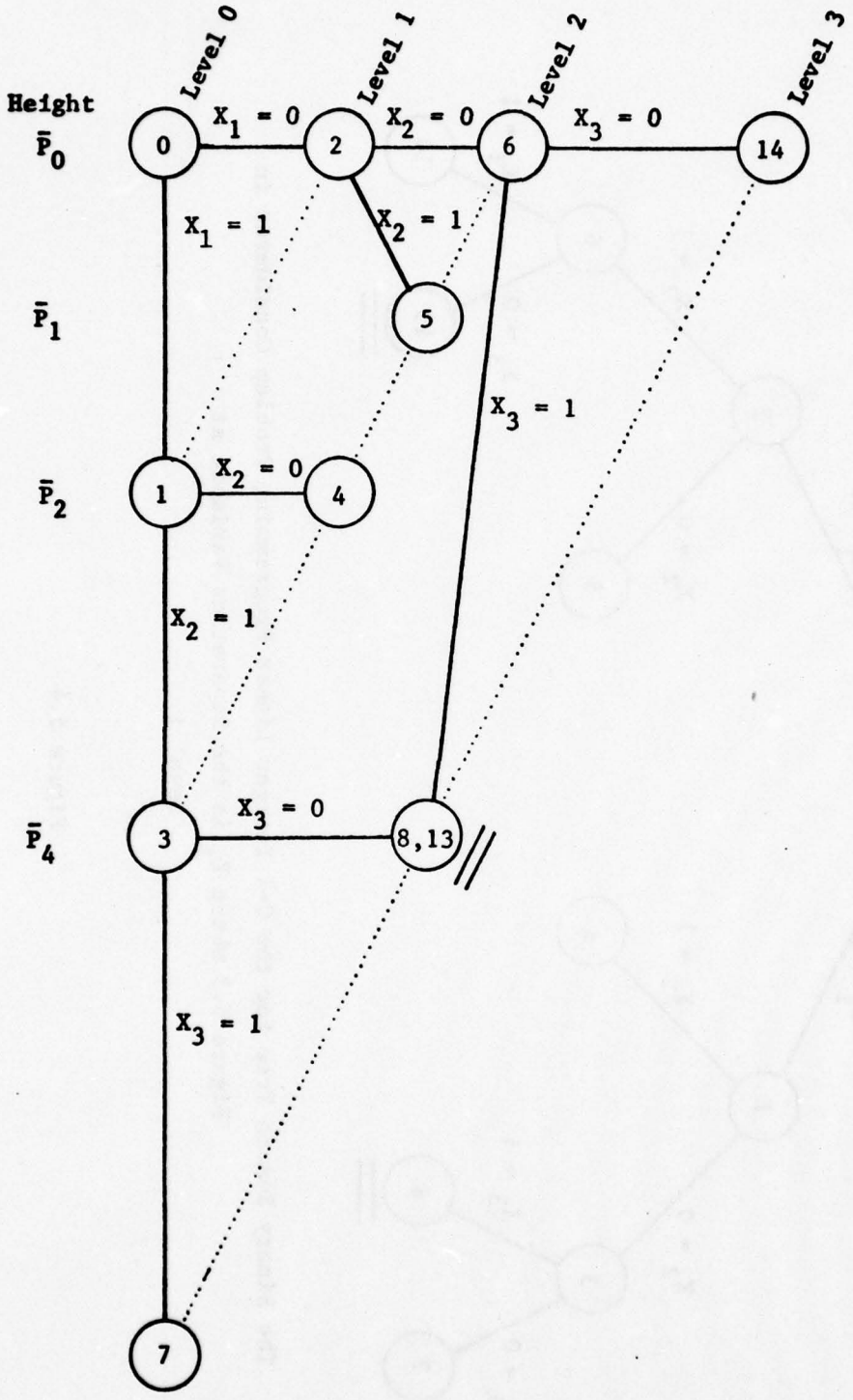
If we were to use the mapping $\bar{P}$, the decision tree of Figure 5.4 could be redrawn as in Figure 5.5, where the height of a node in the decision tree indicates its position in the linear ordering induced by $\bar{P}$. Nodes having the same height can be recognized as having the same resource utilization vector. Nodes at the same height and the same level are equivalent (i.e. they have the same resource utilization and same set of assigned variables). The subsequent development for them need only be done once.

The advantage of the branch and bound procedure is that we are likely to find a feasible solution early on in the search. We can then use this incumbent solution to bound other nodes. Most shortest path algorithms, on the other hand, do not generate a feasible solution until the procedure is very near to its conclusion. Methods have been devised to eliminate this shortcoming



The Binary Search Tree for the 0-1 Integer Linear Programming Problem Considered in Figure 5.3 where X_j is the Separation Variable at Level j

Figure 5.4



The Decision Tree of Figure 5.4 where the Height of a Node Indicates its Position in the Linear Ordering Induced by $\bar{P}$

Figure 5.5

of shortest path algorithms [Shapiro & Wagner (1967), Glover (1967), Shapiro (1968)]. For example, if Glover's algorithm is interpreted in a shortest path context (it is developed as a dynamic programming recursion technique) it can be seen to be equivalent to a shortest path algorithm that bounds any path which becomes longer than the shortest path. With the mapping $\bar{P}$ it is possible to develop an algorithm equivalent to Glover's which will work on any integer programming problem.

5.5 Applications to the Set Partitioning Problem

In Section 5.4 we have indicated how the use of prime numbers might expedite branch and bound procedures for general integer programming problems. For the set partitioning problem, because of its special properties, we can do much better.

The first property of set partitioning problems is that the variables may be constrained to be 0 or 1. In practice, though, it is not necessary to consider the upper bounds since the constraints themselves do not

allow any variable to exceed one. It is true for all 0-1 integer programming problems that partial solutions correspond to subsets of $\{1,2,\ldots,N\}$ where $\bar{X}$ corresponds to $S = \{j:\bar{X}_j = 1\}$. Then if the set partitioning problem is (SP)

$$\text{(SP)} \qquad \text{Min} \quad \sum_{j=1}^{N} C_j X_j$$

$$\text{st} \quad \sum_{j=1}^{N} a_{ij} X_j = 1 \qquad i = 1,\ldots,M$$

$$X_j \geq 0, \text{ Integer} \qquad j = 1,\ldots,N$$

where $a_{ij} = 0,1$, the knifedge representation (KSP) is

$$\text{(KSP)} \qquad \text{Min} \quad \sum_{j=1}^{N} C_j X_j$$

$$\text{st} \quad \sum_{j=1}^{N} \left(\sum_{i=1}^{M} a_{ij} \ln P_i \right) X_j = \sum_{i=1}^{M} \ln P_i$$

$$X_j \geq 0, \text{ Integer} \qquad j = 1,\ldots,N$$

As shown in Section 5.3 (KSP) may be interpreted as a problem of finding a shortest path on a graph G = (V,A)

where $V = \{v: v = \sum_{j \in S} \left(\sum_{i=1}^{M} a_{ij} \ln P_i \right) X_j$ where

$S \subseteq \{1,2,\ldots,N\}$ and $X_j = 0,1\}$

i.e. where $V = \{v: v = \sum_{j \in S} \left(\sum_{i=1}^{M} a_{ij} \ln P_i \right)$ where

$S \subseteq \{1,\ldots,N\}\}$.

Consequently there are at most 2^N nodes or vertices on the Graph G.

Another characteristic is that all of the coefficients of the constraints are either 0 or 1. The right hand side is all ones. Consequently, the only nodes which need be considered are

$$V^* = \{v: v = \sum_{i \in T} \ln P_i \quad \text{where } T \subseteq \{1,\ldots,M\}\}$$

This is because these are the values of v which correspond via the one-to-one mapping P to feasible resource utilizations in Z^M. Consequently, there are at most $\text{Min}\{2^N, 2^M\}$ nodes or vertices on the graph G.

We can make further use of the logical implications of the set partitioning problem. We begin by reformulating the Principle of Optimality in terms of shortest path problems.

Let $\bar{X} = \{X_{j_1}^{v1}, X_{j_2}^{v2}, \ldots, X_{j_k}^{vk}\}$ be an ordered optimal path in the shortest path representation of (KSP), the knifedge reformulation of the set partitioning problem (SP). The $j_1, \ldots, j_k$ are all unique, so we suppress the superscripts v_i with the understanding that each arc X_{j_i} begins where the previous arc $X_{j_{i-1}}$ ends. Then $\bar{X} = \{X_{j_1}, \ldots X_{j_k}\}$. The set $X^* = \{X_j : X_j = 1$ for all $X_j \in \bar{X}$, $X_j = 0$ otherwise$\}$ is an optimal solution to (KSP) and (SP). A SUBPATH $\bar{X}_p$ of $\bar{X}$ is the ordered subset of the first P elements of $\bar{X}$, $\bar{X}_p = \{X_{j_1}, X_{j_2}, \ldots, X_{j_p}\}$, $p \leq k$. Note the elements of $\bar{X}_p$ must retain the order they have in $\bar{X}$. $\bar{X}_p$ corresponds to a partial solution X_p^* of (SP) where $X_p^* = \{X_j : X_j = 1$ for all $X_j \in \bar{X}_p$, X_j undetermined otherwise$\}$.

Principle of Optimality: An ordered optimal path $\bar{X} = \{X_{j_1}, \ldots, X_{j_k}\}$ to a shortest path problem is said to satisfy the Principle of Optimality if and only if any ordered subpath $\bar{X}_p = \{X_{j_1}, \ldots, X_{j_p}\}$, $p \leq k$, has the

property that it is an optimal path for the nodes it covers.

Notice that some ordering of $\{X_{j_1}, X_{j_2}, \ldots, X_{j_k}\}$ is a solution to the shortest path problem which can be generated by usual techniques (e.g. Algorithm 3.2). However, in general it is not the case that an arbitrary permutation of $\{X_{j_1}, X_{j_2}, \ldots, X_{j_k}\}$ will satisfy the principal of optimality. Consider the problem

$$\begin{aligned} \text{Min} \quad & X_1 + X_2 + 3X_3 \\ \text{st} \quad & X_1 + 2X_2 + 3X_3 = 4 \\ & X_j = 0,1 \qquad\qquad j = 1,2,3 \end{aligned}$$

The optimal solution is $X_1 = X_3 = 1$, $X_2 = 0$. The corresponding optimal path to the shortest path interpretation is (X_1, X_3). This permutation satisfies the principle of optimality. However, the permutation (X_3, X_1) does not since (X_3) is not an optimal subpath ((X_1, X_2) is a better subpath). Consequently, it is not the case that an arbitrary permutation can be generated by the usual shortest path algorithms. However, it is the case for set partitioning problems that any

permutation of an optimal answer will satisfy the principle of optimality.

Proposition 5.3: For the set partitioning problem (SP) and the corresponding knifedge problem (KSP), any permutation of an optimal answer will satisfy the principle of optimality with respect to the shortest path interpretation of (KSP).

Proof: Consider a permutation $\bar{X} = \{X_{j_1}, X_{j_2}, \ldots, X_{j_k}\}$ of an optimal solution to (KSP). Consider a subpath $\bar{X}_p = \{X_{j_1}, X_{j_2}, \ldots, X_{j_p}\}$, $p \leq k$, with span P. Suppose $\bar{X}_p$ is not an optimal path with span P, but rather $\bar{Y}_t = \{Y_1, Y_2, \ldots, Y_t\}$ is. Then by Proposition 5.2 and the fact that no row of the original set partitioning problem can be covered more than once, we have, $\bar{Y}_t \cap (\bar{X} \setminus \bar{X}_p) = \emptyset$. But then $\bar{Y}_t \cup (\bar{X} \setminus \bar{X}_p)$ is a better solution than $\bar{X}$ to the original problem, which cannot be the case.

QED

Proposition 5.3 is very powerful. It allows us, in advance, to order the optimal solution in any fashion we choose. We choose to order the arcs of the solution according to the row containing their first entry. Then at any node of our shortest path algorithm we need consider only those arcs which begin at the first uncovered row. And of those arcs we need consider only those that do not conflict with the shortest path to the node being considered. This drastically reduces the number of nodes generated.

Consider the following algorithm to solve a set partitioning problem (SP).

Algorithm 5.1:

STEP 1 From (SP) form the corresponding knifedge problem (KSP). Interpret (KSP) as a shortest path problem where the number of nodes is

$$\sum_{i=1}^{M} \ln P_i$$

where P_i is the ith prime number, and the span of each arc X_j^v is

$$\sum_{i=1}^{M} a_{ij} \ln P_i .$$

STEP 2 Solve the shortest path problem generating nodes as needed. A node is characterized by its span from the origin, which is given by the sum of the spans of the arcs in a path reaching it. At a given node consider only the arcs which begin in the first row of (SP) not yet covered, eliminate those which conflict with the path to date.

We tried implementing Algorithm 5.1 and found that it was very sensitive to problem size. Small set partitioning problems were solved with great speed. A test problem of 30 rows and 168 columns was solved in less than two seconds of execution time on a CYBER 175, generating only 2314 nodes. However, the same problem with ten additional rows and 103 additional columns generated more than 20,000 nodes (in 7 seconds) and was abandoned (it was approximately one-half of the way through the problem).

5.6 Applications to the Helsinki Problem

In conjunction with the HELSINKI algorithm, Algorithm 5.1 is very powerful. The iterative backward-forward shortest path Algorithm 3.3 plus subgradient optimization allows us to eliminate variables at each node which cannot be eliminated by logical considerations alone. Our experience has been that 75% of the variables can be eliminated at the first level of the decision tree (i.e. when only one variable has been chosen to be in the solution). This makes Algorithm 5.1 very attractive as a "relaxation" in the two duty period scheduling problem. Our practice has been to use Algorithm 5.1 when the number of variables remaining in the problem falls below some number (between 60 and 120). Algorithm 5.1 is very useful in cleaning up small problems which very often require a significant number of branches otherwise. Algorithm 5.1 may also be a good relaxation to incorporate into implicit enumeration algorithms for solving general set partitioning problems.

An additional advantage of the Knifedge relaxation is that it makes it very easy to handle certain side

constraints. We incorporated the constraint

$$\sum_{j=1}^{N} X_j \leq K$$

This may be done by adding a slack to get

$$\sum_{j=1}^{N} X_j + S = K$$

multiplying the constraint by $\ln P_{M+1}$ and adding it to the Knifedge constraint for (IP). Then, however, the arc S must be considered at every node and the path is significiantly longer (by $K \ln P_{M+1}$). There is a way in which this constraint can be implemented logically so that the path is shorter rather than longer. At each node Algorithm 5.1 checks to see if a feasible solution has been found by finding the first row not covered. Consequently, whenever a feasible solution is found the path ends there and the solution is recorded if it is the best so far. If the current node is not a feasible solution, that is there is a row < M not yet covered, then the number of arcs in the path to this node is checked. If it is equal to K, the node is fathomed. In this way many more nodes are fathomed along the way and the arc S is never considered.

We have used the Knifedge relaxation in the two duty period scheduling problem and have found that it significantly reduces the running time (5 - 30%). In addition it gives the capability of more easily considering side constraints.

5.7 Applications to the Set Covering Problem

An algorithm similar to Algorithm 5.1 can be developed for the set covering problem (SC):

$$\begin{aligned} (SC) \qquad & \text{Min} \quad CX \\ & \text{st} \quad AX \geq 1 \\ & \qquad X = 0,1 \end{aligned}$$

where $a_{ij} = 0,1$; $C > 0$.

Consider the shortest path interpretation of the knifedge formulation (KSC) of (SC). We show that any permutation of an optimal solution to (SC) will satisfy the principle of optimality.

Proposition 5.4: For the set covering problem (SC) and the corresponding knifedge problem (KSC), any permutation of an optimal answer will satisfy the Principle of Optimality with respect to the shortest path interpretation of (KSC).

Proof: Consider a permutation $\bar{X} = (X_{j_1}, X_{j_2}, \ldots, X_{j_k})$ of an optimal solution to (KSC). Consider a subpath $\bar{X}_p = (X_{j_1}, X_{j_2}, \ldots, X_{j_p})$, $p \leq k$, with span P. Suppose $\bar{X}_p$ is not an optimal path with span P, but rather $\bar{Y}_t = (Y_1, Y_2, \ldots, Y_t)$ is. Let $X_i \in \bar{Y}_t \cap (\bar{X} \backslash \bar{X}_p)$. Then X_i covers rows of (SC) already covered by $\bar{X}_p$ by Proposition 5.2. Then $(\bar{X} \backslash \bar{X}_i)$ is a solution to (SC) better than $\bar{X}$, which cannot be. Therefore, $\bar{Y}_t \cap (\bar{X} \backslash \bar{X}_p) = \emptyset$ which implies $\bar{Y}_t \cup (\bar{X} \backslash \bar{X}_p)$ is a solution to (SC) better than $\bar{X}$, which cannot be.

QED

The following algorithm will solve the set covering problem.

Algorithm 5.2:

STEP 1 From (SC) form the corresponding knifedge problem (KSC). Interpret (KSC) as a shortest path problem where the number of nodes is

$$\sum_{i=1}^{M} \ln P_i$$

where P_i is the ith prime number, and the span of each arc X_j^v is

$$\sum_{i=1}^{M} a_{ij} \ln P_i .$$

STEP 2 Solve the shortest path problem generating nodes as needed. A node is characterized by its span, which is given by the sum of the spans of the arcs in a path reaching it. At a given node consider only the arcs which cover the first row of (SC) not yet covered.

We show that Propositions 5.3 and 5.4 are special cases of a more general result.

Proposition 5.5: For the integer programming problem (IP) with no upper bounds on the variables and the corresponding knifedge problem (KIP), any permutation of an optimal answer will satisfy the principle of optimality with respect to the shortest path interpretation of (KIP).

Proof: Consider a permutation $\bar{X} = (X_{j_1}^{v_1},\ldots,X_{j_k}^{v_k})$ of an optimal solution to (KIP). Consider a subpath $\bar{X}_p = (X_{j_1}^{v_1},\ldots,X_{j_p}^{v_p})$, $p \leq k$, with span P. Suppose $\bar{X}_p$ is not an optimal path with span P, but rather $\bar{Y}_t = (Y_{h_1}^{u_1},\ldots,Y_{h_t}^{u_t})$ is. Then $\bar{Y}_t \cup (\bar{X}\backslash\bar{X}_p)$ is a solution to (KIP) better than $\bar{X}$, which cannot be.

QED

Note that the proof depends upon the variables having no upper bounds, otherwise $\bar{Y}_t \cup (\bar{X}\backslash\bar{X}_p)$ may violate an upper bound on some X_j. The result does not hold generally for problems where the variables are restricted to be 0,1. However, as we have already seen, in the set partitioning problem the variables need not be constrained to be 0,1. Similarly, for the set covering problem with $C > 0$, the variables need not be

bounded above. Consequently, Propositions 5.3 and 5.4 follow immediately from Proposition 5.5.

Similarly, for any 0,1 integer programming problem where the constraints force the 0,1 condition, Proposition 5.5 holds. An example would be a problem where each 0,1 variable is included in a multiple choice constraint of the form

$$\sum_{j \in S} x_j = 1$$

where $S \subseteq \{1,\ldots,N\}$.

CHAPTER 6

THE GENERAL TWO DUTY PERIOD SCHEDULING PROBLEM

The general two duty period scheduling problem (P) is given by

$$
\begin{aligned}
(P) \quad & \text{Min} && CX \\
& \text{st} && AX = b \\
& && X \quad \text{Integer, Nonnegative}
\end{aligned}
$$

where each column of A contains at most two segments of ones, the rest of the entries being zero.

As developed in Chapter 2, (P) may be reformulated as (P*)

$$
\begin{aligned}
(P^*) \quad & \text{Min} && C^Y Y + C^Z Z \\
& \text{st} && (Y,Z) \text{ in } S \\
& && IY - IZ = 0 \\
& && Y,Z \quad \text{Integer, Nonnegative}
\end{aligned}
$$

where S is the set of solutions to a network flow problem.

Much of the methodology developed in chapters three and four may be simply modified to fit the more general two duty period scheduling problem. The relaxation now is a network flow problem rather than the shortest path problem.

For the same reasons that the shortest path was acyclic, the network flow problem will also be acyclic. Consequently, we still have a reasonably easy relaxation to solve. However, in the case of greater than or equal to constraints the network flow problem will no longer be acyclic.

The major drawback with the general two duty period scheduling problem is that we are not able to make such extensive use of logical reduction. Much of the power of the HELSINKI algorithm came from the fact that once an arc was chosen to be in the shortest path, we were able to eliminate all arcs which conflicted with it. In the case of the network problem this is not true. However, some logical reduction may still be useful.

Consider the case where $a_{ij} \geq 0$ for all elements a_{ij} of A_j (i.e. there are no surplus variables). Suppose we have chosen enough arc flows (X_j, j in J) in the network flow problem such that the ith constraint is satisfied, i.e.

$$\sum_{j=1}^{N} a_{ij} X_j = b_i$$

Then we can eliminate (set equal to zero) all other X_j's (j not in J) such that $a_{ij} > 0$. And of course we are still able to eliminate the partner of any eliminated arc.

We can also use subgradient optimization in a completely analogous way. We start by solving the network relaxation. Then by the use of subgradient optimization we reallocate the cost of a variable between its two decoupled arcs in such a way as to make them more nearly equally desirable. This is done by increasing the costs of those arcs which were included in the network solution but whose partners were not included. Simultaneously, we reduce the costs of the unincluded partners by the same amount.

One thing that will have to be done differently with the general two duty period scheduling problem is the branching procedure in the tree search. Rather than using as a separation variable the first arc in the shortest path relaxation solution not already used as a separation variable, we will have tc choose a branching procedure adapted to the network relaxation.

CHAPTER 7

COMPUTATIONAL EXPERIENCE

In this chapter we present our computational experience with the HELSINKI algorithm. In Section 7.1 we discuss the test problems used. We follow this with results on the logical elimination of variables and results of the subgradient optimization procedure. In Sections 7.4 and 7.5 we present computation times for the test problems with and without a side constraint. Decision trees for the two smaller test problems are given in Section 7.7. Section 7.8 provides some results of the prime number - shortest path algorithm for small test problems.

7.1 The Test Problems

Our computational experience has dealt with the Helsinki problem discussed in Chapter 3:

$$
\begin{aligned}
\text{(HP)} \quad & \text{Min} \quad CX \\
& \text{st} \quad AX = 1 \\
& \qquad X = 0,1
\end{aligned}
$$

where each column of A contains at most two segments of ones, the rest of the entries being zero. (HP) is a two duty period scheduling problem. It is also a set partitioning problem and, consequently, good methods already exist for solving it. Our intention was to test our method for solving the two duty period scheduling problem by considering first the Helsinki problem. If we were able to obtain results comparable to the best existing set partitioning algorithms, then Algorithm 2.1 should provide a reasonable method for solving the more general two duty period scheduling problem

$$
\begin{aligned}
\text{(P)} \quad & \text{Min} \quad CX \\
& \text{st} \quad AX = b \\
& \qquad X \quad \text{Integer, Nonnegative}
\end{aligned}
$$

for which no good solution techniques exist.

The test problems used in evaluating our HELSINKI algorithm were sent to us by Markku Tamminen of the

Helsinki City Transport. The four test problems have roughly the same size but varying densities. See Table 7.1.

Problem 1 was the original Helsinki test problem and has been studied rather thoroughly by us and others. Tamminen tested a number of set partitioning algorithms on problem 1 [Koljonen & Tamminen (1977)]. These included the Garfinkel-Nemhauser algorithm and Marsten's set partitioning algorithm, SETPAR. In addition Tamminen tested numerous variations of the Garfinkel-Nemhauser algorithm, with and without linear programming and taking advantage of the special two segment structure of the problem. His conclusion was that SETPAR outperformed any other method they could devise. Throughout this chapter our results with the HELSINKI algorithm will be compared with SETPAR, it being the best known existing algorithm for solving the Helsinki problem.

Problem	Size	Density	LP	SG	IP	$\frac{IP-LP}{LP}$	$\frac{LP-SG}{SG}$	HELSINKI	SETPAR
1	57 X 551	11.303%	696.6	669	802	15.1%	4.1%	26.336	71.211
2	78 X 525	7.062%	1576.8	1555	1637	3.8%	1.4%	87.679	(84.334)*
3	73 X 412	7.302%	887.6	863	1294	45.8%	2.8%	174.303	
4	60 X 411	9.444%	457.7	449	511	11.6%	1.9%	15.711	50.044

* Problem abandoned, best solution found was 1893

Size: Rows X Columns
LP: Linear Programming relaxation optimal value
SG: Best value obtained by subgradient optimization of the Lagrangean relaxation, level 0
IP: Integer Program optimal value
$\frac{LP-SG}{SG}$: Gap between LP and SG as percentage of SG
$\frac{IP-LP}{LP}$: Gap between IP and LP as percentage of LP
HELSINKI: Solution time for HELSINKI algorithm, CPU seconds on a CYBER 175 using FORTRAN IV
SETPAR: Solution time for SETPAR algorithm, CPU seconds on a CYBER 175 using FORTRAN IV

Computational Data for the Four Test Problems

Table 7.1

7.2 Effectiveness of Logical Reduction in the HELSINKI Algorithm

One of the main features of the HELSINKI algorithm is its reliance on logic to reduce the problem size. This logical reduction entails two features. The first is to eliminate any variables which conflict with variables already chosen to be in the solution. The second is to eliminate any variables which can not participate in a solution improving upon the best known solution (the incumbent).

Our experience is that both of these features are instrumental in significantly reducing the problem size. Table 7.2 indicates the reductions we have experienced.

Table 7.2 does not accurately reflect the number of variables which could be eliminated at each level. As soon as a branch was bounded or found to be infeasible, no more variables were eliminated. In those cases one hundred per cent of the variables could have been eliminated.

Level	1	2	3	4	Average
0	3.8	4.2	5.1	1.2	3.6
1a	42.6	39.0	27.4	20.4	32.4
1b	89.8	70.5	62.6	81.5	76.1
2	91.6	83.5	75.5	81.9	83.1
3	90.2	87.6	79.8	81.7	84.8
4	88.6	87.7	81.1	63.7	80.3

Percent of Variables Eliminated at each

Level of the Decision Tree

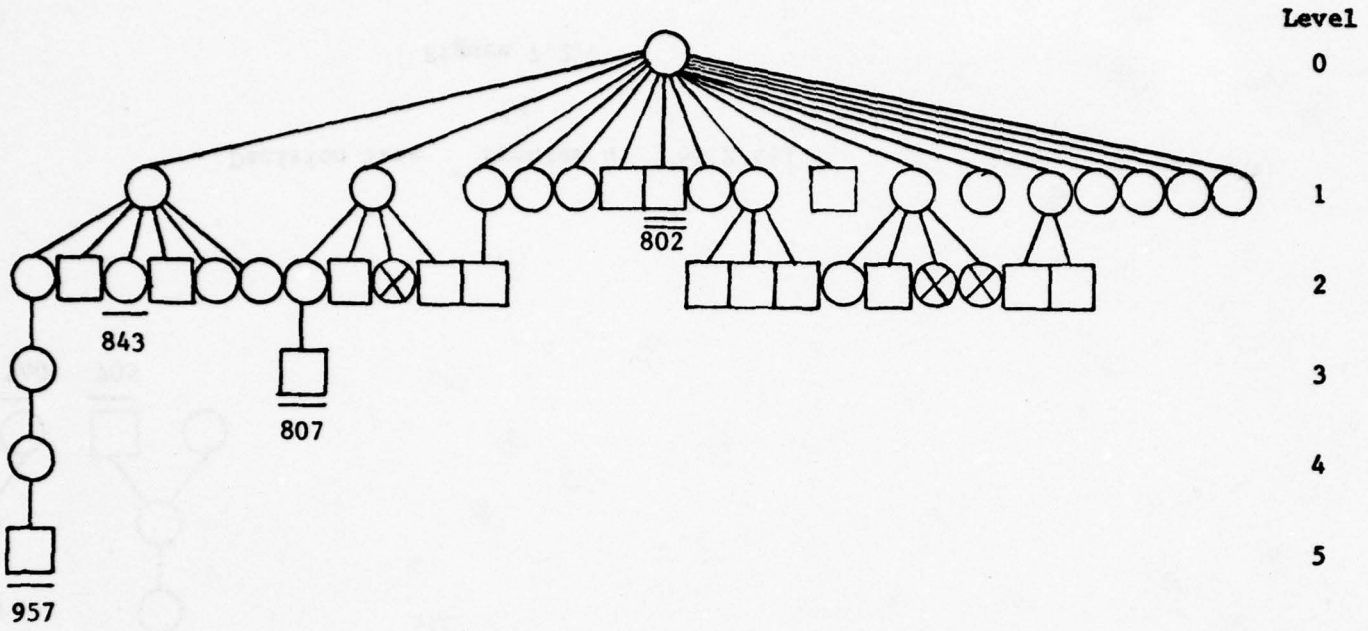
Table 7.2

At level one we have recorded the percentage of variables for two cases. The first case, 1a, is the first branch at level 1 (when there was no incumbent solution) and the second case, 1b, is for subsequent branches (when there was an incumbent solution). The difference in these two percentages indicates the additional power gained from the elimination of variables which cannot take part in a solution better than the incumbent. This elimination was achieved by the forward-backward iterative solution process for the shortest path problem as described in Chapter 3, Algorithm 3.3.

Note that, for problems 1 and 4, fewer variables are eliminated at level 4 than at level 3. This apparent anomaly is due to the fact that level 4 was visited only when there was no incumbent solution or only a weak incumbent (see Figures 7.1 and 7.2).

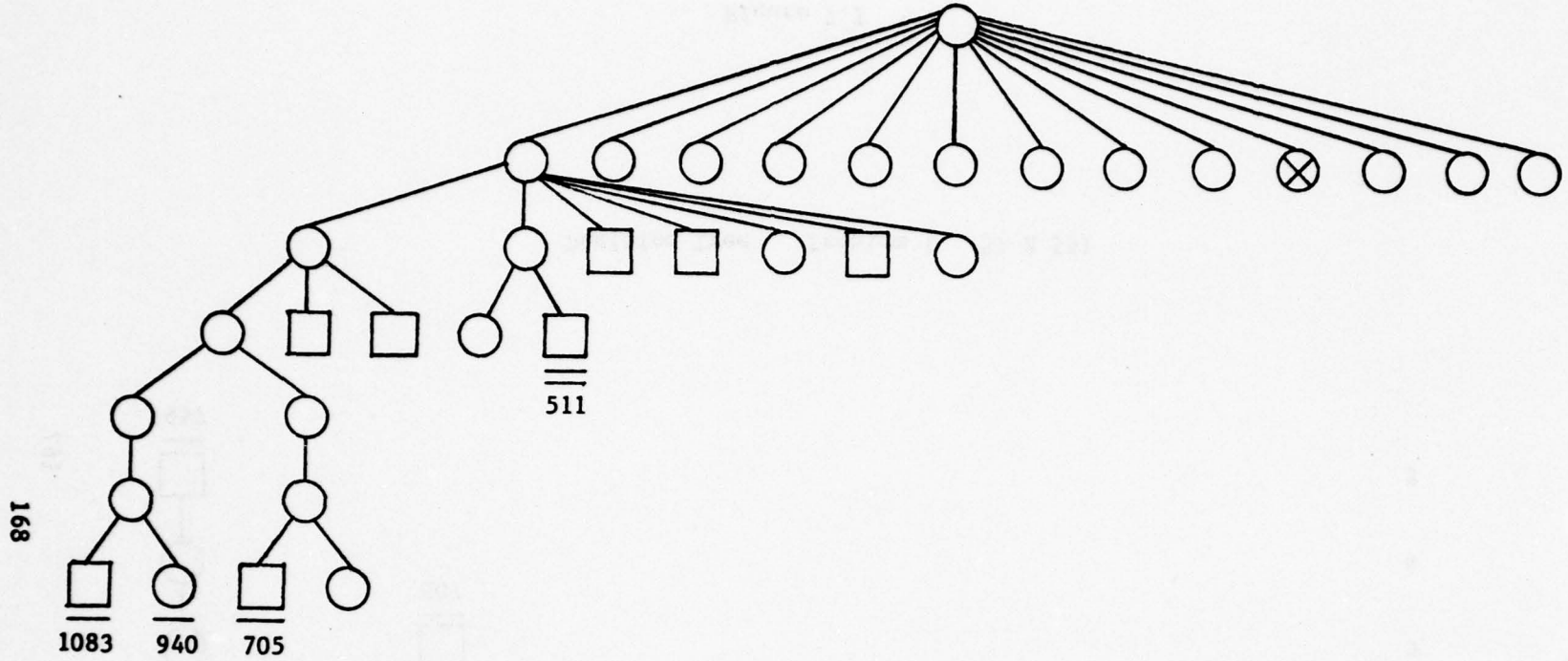
7.3 Results of the Subgradient Optimization Procedure

We found the subgradient optimization procedure to be very successful in tightening our relaxation. Recall



Decision Tree Problem 1 57 X 551

Figure 7.1



Decision Tree Problem 4 60 X 411

Figure 7.2

that the Lagrangean relaxation of the Helsinki problem has the integrality property. Therefore, we know that the subgradient optimization procedure can be expected to converge to the linear programming relaxation solution. In practice we were able to approximate the linear programming solution as closely as we wished by simply increasing the number of iterations in the subgradient optimization procedure. The most difficult aspect was determining the best number of iterations. We finally settled on a number of iterations (about 600 at level 0) which resulted in a relaxation value within 5% of the linear programming value (see Table 7.1). We found that advantages gained by tightening the relaxation even further were more than offset by increased computation times in performing the additional iterations needed.

As outlined in Chapter 4, at each iteration of the subgradient optimization we determine a new set of costs for the decoupled arcs and then resolve the shortest path relaxation. The costs C_j^{t+1} at iteration t + 1 are computed by the following formula

$$C_j^{t+1} = C_j^t + p^t$$

where

$p^t = 1$ if arc j is included in the current solution and its partner is not

$p^t = -1$ if arc j is not included in the current solution and its partner is

$p^t = 0$ otherwise

p^t was determined according to the formula

$$p^t = d^t [\hat{W} - W(U^t)] / |s^t|^2$$

There are two degrees of freedom in determining p^t. The first is the choice of the multiplier d^t, and the second is the choice of the target value $\hat{W}$. We have tried a number of procedures for choosing the target value. Our final algorithm incorporates many of them.

We have found that at the very top of the decision tree, before any subgradient optimization has been done, the shortest path relaxation is very weak. Characteristically, it has been on the order of twenty-five percent of the linear programming value and twenty-two percent of the optimal value. If we were to use the optimal answer itself as the target, the perturbation term

$(d^t[\hat{W} - W(U^t)]/|s^t|^2)$ in the equation above would be very large. Our experience in this case is that the subgradient optimization performs very poorly until the multiplier d^t is adjusted to compensate for the large $[\hat{W} - W(U^t)]$. Consequently, at the top of the tree we use for the target value two times the best relaxation value found so far. Similarly, at lower levels of the tree, until an incumbent solution is obtained, we use 1.2 times the best relaxation value found so far at that branch.

A second alternative is for the user to set the target value. In many real world implementations it turns out that the user has a very accurate notion of what the optimal value will be. In such situations it seems best to take advantage of such knowledge. After trying a number of alternatives, we have settled for using such a guess, if provided, only after the first level of the decision tree.

A third alternative is to take advantage of the relaxation value obtained at level 0 to set targets at lower levels. This value closely approximates the linear programming value. There is wide experience

concerning the gap between the linear programming solution and the integer solution. (Our experience with these four test problems is that they have abnormally large gaps - see Table 7.1.) It is possible to use the relaxation value at level 0 of the tree in order to approximate the value of the integer solution. This approximation could then be used as the target value for the subgradient optimization. In practice, the user has the option of providing a Optimum/Relaxation ratio which the HELSINKI algorithm will use to determine a target value for the subgradient optimization procedure after level 0. If the user guesses this ratio to be, say, 1.30, the HELSINKI algorithm will multiply the relaxation value, obtained by the subgradient optimization procedure at level 0, by 1.30. The resulting value will be used as a target value for subsequent subgradient optimizations and as an upper bound on the problem. Once a solution better than this target value is found it is subsequently used as the target and upper bound.

As the value of the Lagrangean approaches the target value the perturbation term $d^t[\hat{W} - W(U^t)]/|s^t|^2$ becomes very small. It is, therefore, very difficult

for the Lagrangean to exceed the target. Consequently, rather than using the incumbent, the best known solution to the problem, as a target value; we have made a practice of using 1.2 times the incumbent for the target. We have found this to be very effective.

The second degree of freedom in determining P^t is the choice of the multiplier d^t. The theoretical restrictions on d^t are that at each iteration d^t be between 0 and 2, that the sum of the d^t's must diverge and that d^t must go to 0 in the limit [Held, Wolfe & Crowder (1974)]. Our practice has been to start with d^t large, do a number of iterations and then reduce the size of d^t. We continue in this manner until either the perturbation term goes to zero or we have reached a preset number of iterations. In practice, we do L iterations with d^t, then set $L = L/2$ and $d^t = d^t/2$ and proceed. Both the initial L and the initial d^t depend on the current level of the tree (see Table 7.3). The subgradient process terminates when d^t is less than the minimum d^t for that level. Below level 2 the parameters are set as they are for level 2.

Level	Initial L	Minimum L	Initial d^o	Minimum d^t
0	320	10	2.0	0.0078125
1	80	10	1.0	0.03125
2	40	10	0.25	0.03125

L is the number of iterations done with the current value of the multiplier d^t.

Parameter Values for the Subgradient Optimization

Table 7.3

7.4 Computation Times for Four Test Problems

Table 7.1 indicates run times for the four problems under the following conditions. No side constraints were considered. No guess was admitted. No Optimum/Relaxation ratio was given. In other words Table 7.1 gives the results for the HELSINKI algorithm when the straight Helsinki problem was considered with no outside information. Included in the table are the results for the set partitioning algorithm SETPAR under the same conditions. CPU seconds are on a CYBER 175 computer using FORTRAN IV.

Table 7.4 indicates run time for both algorithms incorporating a Optimum/Relaxation ratio. It should be noted that since the two algorithms use different methods (subgradient optimization and linear programming) to get relaxation values at the top of the decision tree the same ratio will lead to different upper bounds for the two, where upper bound equals ratio times relaxation value. (Recall that the subgradient procedure stops short of optimality.) Consequently, we have chosen ratios for the HELSINKI algorithm to give the same resulting upper bound that SETPAR calculated.

Problem Number	SETPAR Ratio	Resulting Upper Bound	HELSINKI Run Time	SETPAR Run Time	Problem Feasible?
2	1.10	1734.5	88.824	(116.642)*	YES
3	1.15	1020.7	9.368	20.557	NO
3	1.50	1331.3		(116.095)†	YES
4	1.10	503.4	10.250	13.230	NO
4	1.25	572.1	13.120	14.670	YES

* No feasible solutions found, problem abandoned.

† Problem abandoned, best solution found was 1484

Problem Feasible?: Were there any feasible solutions with a value less than the Resulting Upper Bound?

Results with No Side Constraints, No Guess, but Optimum/Relaxation Ratio Given

Table 7.4

7.5 Results when Side Constraints are Included

The HELSINKI algorithm may be modified to handle side constraints. A common side constraint in crew scheduling problems is that the number of variables in the solution be less than or equal to some number K.

$$\sum_{j=1}^{N} X_j \leq K$$

As mentioned in Chapter 5, the HELSINKI algorithm can take advantage of this side constraint in its prime number - shortest path relaxation. However, the HELSINKI algorithm utilizes an additional technique to include this side constraint. At each node of the branch and bound search tree, there are now two phases. In phase one we consider a new problem formulated by replacing the objective function of the original Helsinki problem with our new constraint to form the new objective function

$$\text{Min} \quad \sum_{j=1}^{N} X_j$$

We use the same techniques developed in Chapters 2 through 4 to solve this new problem. We form the

Lagrangean relaxation and use subgradient optimization to tighten that relaxation. Using the backward-forward Algorithm 3.3 to evaluate the relaxation, we are able to eliminate many arcs which cannot participate in a solution containing at most K variables. If we fail to fathom the node using the side constraint as the objective function, we then proceed to phase two. The second phase entails solving the regular subproblem (with the original objective function). The elimination of variables and their partners in phase one strengthens our relaxation in phase two.

Both SETPAR and HELSINKI are capable of handling the above restriction on the number of variables in the optimal solution. Table 7.5 compares the two algorithms for problems with this side constraint.

7.6 Decision Trees for the Four Test Problems

In order to give an impression of the actual computation of the algorithm in solving the four test problems, we include the decision trees generated in problems 1 and 4. See Figures 7.1 and 7.2. Some

Problem Number	K	HELSINKI Run Time	SETPAR Run Time	Problem Feasible?
1	9	2.440		NO
2	16	2.652		NO
4	10	1.187	10.006	NO
4	11	18.131	16.451	YES

Problem Feasible?: Were there any feasible solutions with no more than K nonzero variables?

Results with No Guess, No Optimum/Relaxation Ratio but Side Constraint $\sum_{j=1}^{N} X_j \leq K$ Included

Table 7.5

special notation has been employed. Nodes with an X through them, ⊗ , indicate nodes that were fathomed when the logical reduction techniques demonstrated that the problem was infeasible at that node. Nodes where a feasible solution was found are underlined and the value of the feasible solution is entered. The node where the optimal solution was found is underlined twice. All other hanging nodes were fathomed by bounding. Those nodes which were fathomed using the prime number - shortest path algorithm are depicted by squares rather than circles.

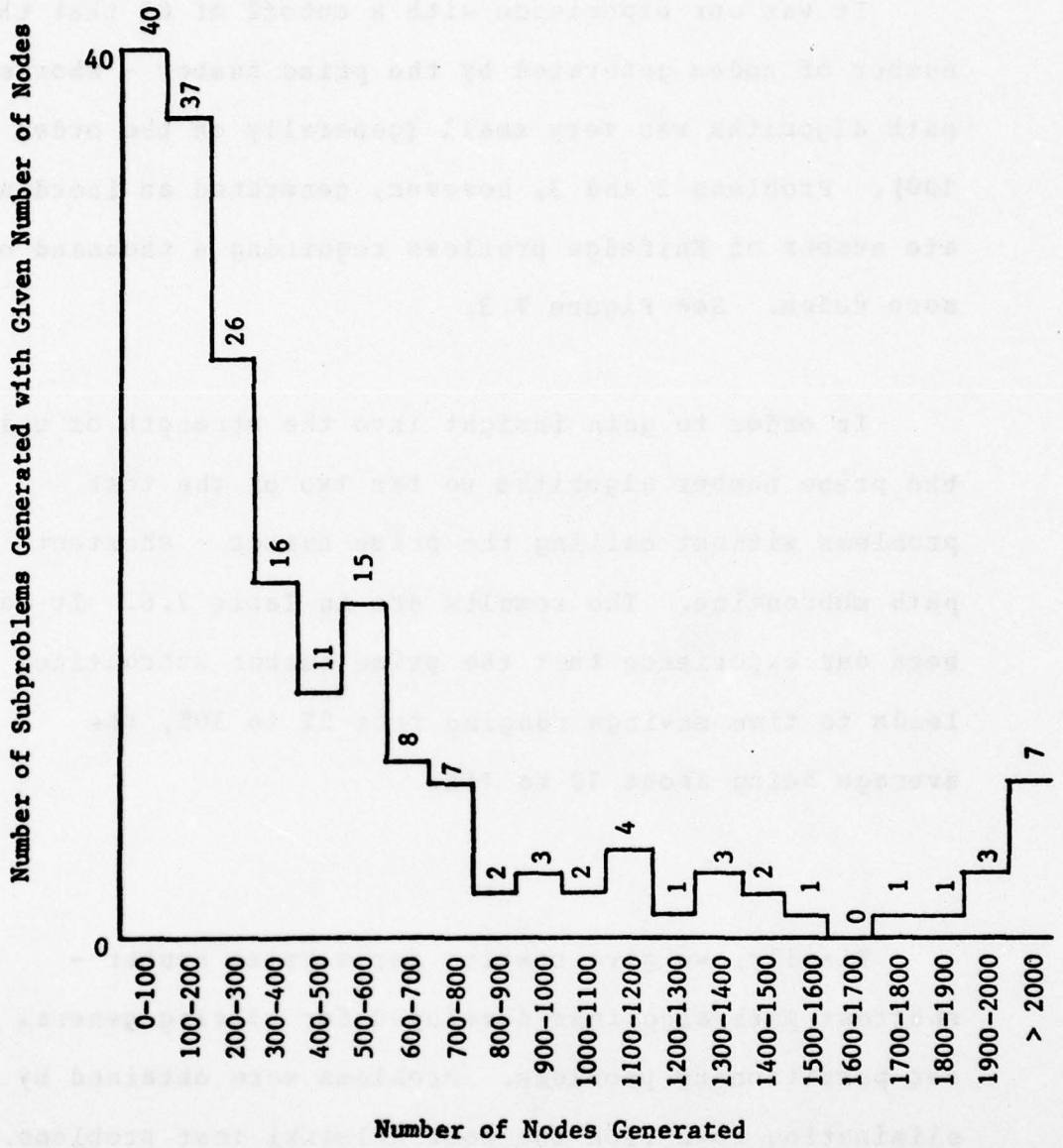
7.7 Results of the Prime Number - Shortest Path Algorithm

For the runs reported here the prime number - shortest path subroutine was called whenever the number of active variables became less than 60. We have also made runs using a cut off of up to 125 variables. We have found 75 to be most effective for easy problems (numbers 1 and 4) and 60 to be most effective for the two hard problems.

It was our experience with a cutoff of 60 that the number of nodes generated by the prime number - shortest path algorithm was very small (generally on the order of 100). Problems 2 and 3, however, generated an inordinate number of Knifedge problems requiring a thousand or more nodes. See Figure 7.3.

In order to gain insight into the strength of using the prime number algorithm we ran two of the test problems without calling the prime number - shortest path subroutine. The results are in Table 7.6. It has been our experience that the prime number subroutine leads to time savings ranging from 5% to 30%, the average being about 10 to 15%.

Finally, we give results for a prime number - shortest path algorithm developed for solving general set partitioning problems. Problems were obtained by eliminating rows from the four Helsinki test problems. See Table 7.7.



Number of Nodes Generated in Prime Number - Shortest Path Subproblems

Figure 7.3

Problem	Threshold 60	0
1	26.336	30.231
4	15.711	16.398

Threshold: When the number of variables in a subproblem is less than or equal to the threshold, the prime number - shortest path subroutine is called.

Run times are CPU seconds on a CYBER 175 using FORTRAN IV.

Run Times for the HELSINKI Algorithm with Different Threshold Values for Calling the Prime Number - Shortest Path Subroutine

Table 7.6

Derivation	Size	Nodes	Time
1	20 X 85	114	0.698
2	20 X 39	60	0.569
3	20 X 41	44	0.444
4	20 X 46	147	0.486
1	30 X 168	2314	1.651
2	30 X 79	767	0.901
3	30 X 89	1076	0.873
4	30 X 90	2096	1.256
1	40 X 271	>20,000	>7.056
2	40 X 147	10,745	6.551
3	40 X 139	15,100	9.699
4	40 X 169	>20,000	>7.562

Size: Rows X Columns

Derivation: Number of test problem which was truncated to form this problem

Nodes: Number of nodes generated

Time: Solution time in CPU seconds on a CYBER 175

Results for the Prime Number - Shortest Path Set Partitioning Algorithm

Table 7.7

PART III Extensions and Discussion

CHAPTER 8

THE CIRCULAR ONES PROBLEM

In this chapter we consider the circular ones or cyclical staffing problem. This is the problem of scheduling workers in a planning horizon that has a cyclical nature. We consider the circular ones set partitioning problem. We show that it may be solved by solving K shortest path problems where K is less than the number of rows or columns in the original problem. We interpret the circular ones problem as one of trying to find the shortest path around a circle. In the final section we consider networks on a circle.

8.1 The Circular Ones Problem

The circular ones problem is a special case of the two duty period scheduling problem that has been studied in some detail [Tucker (1971), Tibrewala, Philippe & Browne (1972), Baker (1974), Brownell & Lowerre (1976), Bartholdi, Orlin & Ratliff (1977), Bartholdi & Ratliff (1977)]. The model represents continuous workshifts in cyclical time. That is, each person works a single duty period with no break. By cyclical time we mean that the planning horizon is of a definite duration having a cyclical nature (a day, week, etc.); however, the time chosen to demarcate the beginning and end of the planning horizon is arbitrary. Take for example, the problem of determining the daily work schedule for a continuous operation, see Figure 8.1. The duration is determined, twenty-four hours. However, saying that a day begins and ends at midnight is arbitrary. Generally a single duty period corresponds to a single segment in the column. However, since we are now dealing with cyclical time, a duty period may extend from the night of one day to the morning of the next. See, for example, X_3 in Figure 8.1.

Min $\sum_{j=1}^{7} c_j X_j$

st	X_1	X_2	X_3	X_4	X_5	X_6	X_7		RHS
12 pm - 2 am	0	0	1	1	0	1	0	=	3
2 am - 4 am	0	1	1	1	0	0	0	=	2
4 am - 6 am	0	1	1	0	0	0	0	=	1
6 am - 8 am	0	1	1	0	0	0	0	=	1
8 am - 10 am	0	1	0	0	1	0	0	=	2
10 am - 12 m	0	1	0	0	1	0	0	=	1
12 m - 2 pm	0	0	0	0	1	0	1	=	2
2 pm - 4 pm	0	0	0	0	1	0	1	=	1
4 pm - 6 pm	1	0	0	0	0	0	1	=	1
6 pm - 8 pm	1	0	0	0	0	0	1	=	1
8 pm - 10 pm	1	0	1	0	0	0	0	=	1
10 pm - 12 pm	1	0	1	1	0	0	0	=	2

An Example of the General Circular Ones - Cyclic Staffing Problem

Figure 8.1

Consequently, the circular ones problem is a special case of the two duty period scheduling problem. Indeed, one could expect most of the columns to have only one segment. In this case the network relaxation would be strong and the subgradient optimization could be expected to converge rapidly.

8.2 The Circular Ones Set Partitioning Problem

When the circular ones problem is a set partitioning problem (P)

$$
\begin{array}{lll}
\text{(P)} & \text{Min} & CX \\
 & \text{st} & AX = 1 \\
 & & X = 0,1
\end{array}
$$

the HELSINKI algorithm would be expected to find an optimal solution quickly. However, the circular ones set partitioning problem can be solved more easily. It requires solving only K shortest path problems where K is the greatest number of nonzero entries in a column or the smallest number of nonzero entries in a row, whichever is less.

To show this, consider the solution to the circular ones set partitioning problem. Recall that each column of A represents a CONTINUOUS SHIFT in cyclical time, corresponding to one or two segments of ones in the column. Choose a variable, X_j, in the solution. Let I be the first row of the continuous duty period of A_j (i.e. choose I such that $a_{Ij} = 1$, $a_{I-1,j} = 0$; or else I = 1). Then reorder the rows in the problem according to the transformation Q_I:

$$Q_I(i) = i - I + 1 \qquad i \geq I$$
$$= M - I + i + 1 \qquad i < I$$

Q_I has the effect of rolling the constraint matrix around, in the sense that row I becomes row 1 and the rows retain their same sequential ordering, where row 1 follows the last row, row M. This is equivalent to starting 24 hour days at some arbitrary time, say 4 am, and finishing them at the same time the following day. Under this transformation the columns will still have the circular ones property. Consider the problem created by eliminating all columns having more than one segment after this transformation. This new problem is simply a one duty period set partitioning problem which

we have shown in Chapter 2 to be equivalent to a shortest path problem. Moreover, the optimal solution to the original circular ones set partitioning problem will be an optimal solution to this new problem.

Let K be the maximal number of nonzero entries in a column of the original circular ones set partitioning problem. Then, if we consider any K consecutive rows, any feasible solution to the circular ones problem must contain a continuous duty period which begins in one of those K rows. To show this, consider the first of the K rows, I. Row I is covered by some variable X_j of the feasible solution. If the continuous duty period of X_j starts in row I, we are finished. If not, let k be the last row of the continuous duty period of X_j. Then $Q_I(k) < K$. Otherwise, X_j contains more than K nonzero entries. Then row k + 1 is covered by a continuous duty period which begins in that row, where $M + 1 \equiv 1$.

Consider now the following algorithm.

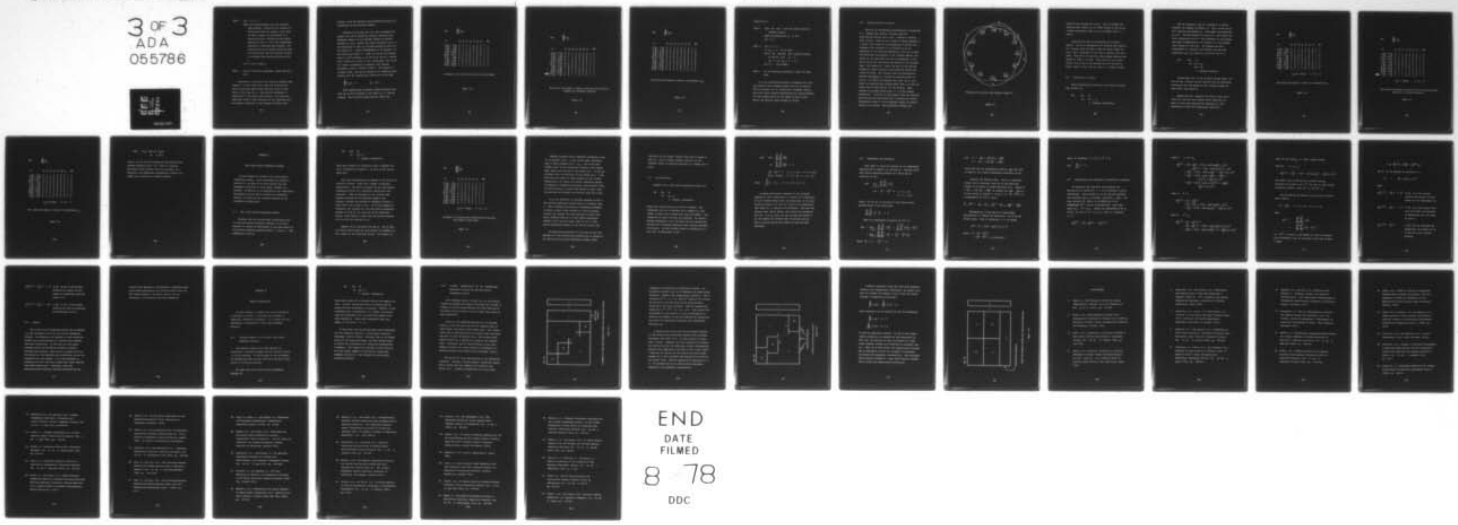
Algorithm 8.1

STEP 0 Let I = 0

AD-A055 786 MASSACHUSETTS INST OF TECH CAMBRIDGE OPERATIONS RESE--ETC F/G 12/1
A LAGRANGEAN RELAXATION ALGORITHM FOR THE TWO DUTY PERIOD SCHED--ETC(U)
JUN 78 W B SHEPARDSON DAAG29-76-C-0064

UNCLASSIFIED TR-152 ARO-14261.9-M NL

3 OF 3
ADA
055786



END
DATE
FILMED
8 78
DDC

STEP 1 Let $I = I + 1$

Apply the transformation Q_I to the circular ones problem. Eliminate all columns now having more than one segment, since none of these columns can participate in a solution having a continuous duty period starting in row I. Solve the transformed problem as a shortest path problem. The solution will be an optimal solution under the restriction that the solution contain a continuous duty period starting in row I.

If $I < K$ go to STEP 1.

STEP 2 Of the K solutions generated, choose the best.

STOP

Algorithm 8.1 will solve the original circular ones problem. We have shown this since the solution will have a continuous duty period starting in one of the first K rows, say row I. Then the Ith iteration of Algorithm 8.1 will find that solution. Of course the solutions found at each iteration of the algorithm will be feasible solutions to the original circular ones

problem, since the shortest path problem generated is a restriction of the original problem.

Likewise we can show that (P) (for an example see Figure 8.2) can be solved by solving K shortest path problems, where K is the smallest number of nonzero entries in a row. Repeat the first constraint as an M + 1st constraint to give an enlarged constraint matrix A'. See Figure 8.3. Apply transformation T of Chapter 2 to the constraint matrix A'. Then TA' can be partitioned into $[a_1 | (TA')^*]$ where a_1 is the first row of A (recall that T leaves the first row of A unchanged), and $(TA')^*$ is a matrix representing a shortest path problem [Bartholdi, Orlin & Ratliff (1977)]. See Figure 8.4. In other words, (P) may be rewritten as a shortest path problem with one complicating constraint of the form

$$\sum_{j=1}^{N} a_{1j} X_j = 1 \qquad a_{1j} = 0,1$$

This complicating constraint simply requires that only one of the variables X_j for which $a_{1j} = 1$ can be nonzero. The following algorithm will solve (P).

$$\text{Min} \quad \sum_{j=1}^{7} C_j X_j$$

st	X_1	X_2	X_3	X_4	X_5	X_6	X_7		RHS
12 pm - 2 am	0	0	1	1	0	1	0	=	1
2 am - 4 am	0	1	1	1	0	0	0	=	1
4 am - 6 am	0	1	1	0	0	0	0	=	1
6 am - 8 am	0	1	1	0	0	0	0	=	1
8 am - 10 am	0	1	0	0	1	0	0	=	1
10 am - 12 m	0	1	0	0	1	0	0	=	1
12 m - 2 pm	0	0	0	0	1	0	1	=	1
2 pm - 4 pm	0	0	0	0	1	0	1	=	1
4 pm - 6 pm	1	0	0	0	0	0	1	=	1
6 pm - 8 pm	1	0	0	0	0	0	1	=	1
8 pm - 10 pm	1	0	1	0	0	0	0	=	1
10 pm - 12 pm	1	0	1	1	0	0	0	=	1

An Example of the Circular Ones Set Partitioning Problem

Figure 8.2

$$\text{Min} \quad \sum_{j=1}^{N} C_j X_j$$

st	X_1	X_2	X_3	X_4	X_5	X_6	X_7		RHS
12 pm - 2 am	0	0	1	1	0	1	0	=	1
2 am - 4 am	0	1	1	1	0	0	0	=	1
4 am - 6 am	0	1	1	0	0	0	0	=	1
6 am - 8 am	0	1	1	0	0	0	0	=	1
8 am - 10 am	0	1	0	0	1	0	0	=	1
10 am - 12 m	0	1	0	0	1	0	0	=	1
12 m - 2 pm	0	0	0	0	1	0	1	=	1
2 pm - 4 pm	0	0	0	0	1	0	1	=	1
4 pm - 6 pm	1	0	0	0	0	0	1	=	1
6 pm - 8 pm	1	0	0	0	0	0	1	=	1
8 pm - 10 pm	1	0	1	0	0	0	0	=	1
10 pm - 12 pm	1	0	1	1	0	0	0	=	1
12 pm - 2 am	0	0	1	1	0	1	0	=	1

The Circular Ones Example of Figure 8.2 with the First Constraint Repeated as a Thirteenth Constraint

Figure 8.3

$$\text{Min} \quad \sum_{j=1}^{N} C_j X_j$$

st	X_1	X_2	X_3	X_4	X_5	X_6	X_7		RHS
12 pm - 2 am	0	0	1	1	0	1	0	=	1
2 am - 4 am	0	1	0	0	0	-1	0	=	0
4 am - 6 am	0	0	0	-1	0	0	0	=	0
6 am - 8 am	0	0	0	0	0	0	0	=	0
8 am - 10 am	0	0	-1	0	1	0	0	=	0
10 am - 12 m	0	0	0	0	0	0	0	=	0
12 m - 2 pm	0	-1	0	0	0	0	1	=	0
2 pm - 4 pm	0	0	0	0	0	0	0	=	0
4 pm - 6 pm	1	0	0	0	-1	0	0	=	0
6 pm - 8 pm	0	0	0	0	0	0	0	=	0
8 pm - 10 pm	0	0	1	0	0	0	-1	=	0
10 pm - 12 pm	0	0	0	1	0	0	0	=	0
12 pm - 2 am	-1	0	0	0	0	1	0	=	-1

The Circular Ones Example of Figure 8.3 Transformed by T_{13}

Figure 8.4

Algorithm 8.2

STEP 0 Find the row I with the least number of nonzero entries.

Apply transformation Q_I to (P).

Let $j = 0$.

STEP 1 Let $j = j + 1$

If $a_{1j} = 0$ GO TO STEP 1

Solve the shortest path problem defined by $(TA')^*$ with $X_j = 1$,

$X_k = 0$ if $a_{1k} = 1$, $k \neq j$

If $j < N$ GO TO STEP 1

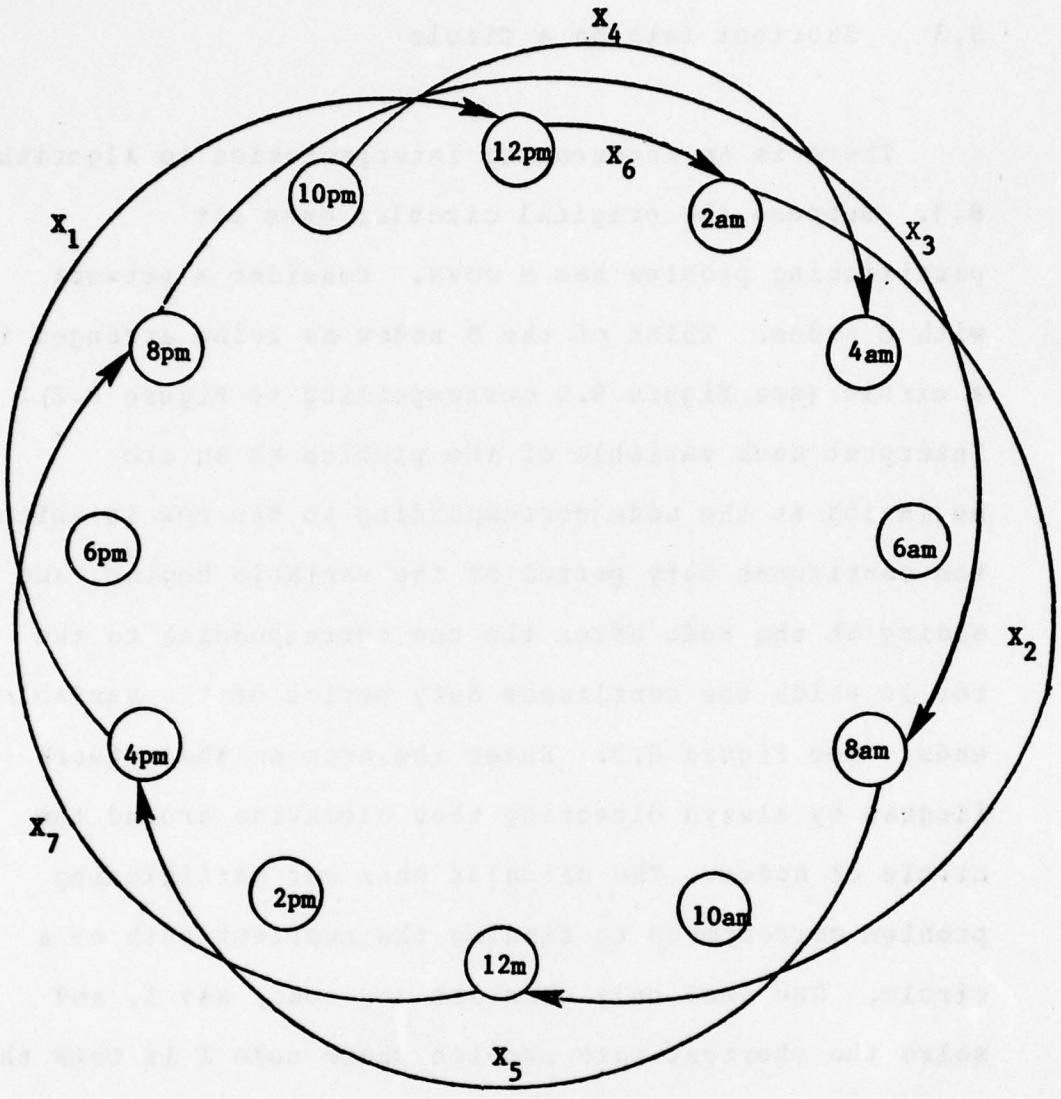
STEP 2 Of the solutions generated, choose the best.

STOP

It is an interesting feature of Algorithm 8.2 that the shortest path problems solved will all be acyclic. This is because, in the transformed constraint matrix, precisely those columns representing arcs going backward are the columns which are set equal to zero or one before the shortest path problem is solved.

8.3 Shortest Path on a Circle

There is an interesting interpretation to Algorithm 8.1. Suppose the original circular ones set partitioning problem has M rows. Consider a network with M nodes. Think of the M nodes as being arranged in a circle (see Figure 8.5 corresponding to Figure 8.2). Interpret each variable of the problem as an arc beginning at the node corresponding to the row in which the continuous duty period of the variable begins, and ending at the node after the one corresponding to the row in which the continuous duty period of the variable ends. See Figure 8.5. Enter the arcs on the network diagram by always directing them clockwise around the circle of nodes. The circular ones set partitioning problem corresponds to finding the shortest path on a circle. One need only consider any node, say I, and solve the shortest path problem where node I is both the origin and the destination for the problem. Then resolve the problem using node I + 1 as the origin/destination. Continue in this manner until the shortest path problem has been solved for K consecutive origin/destinations where K is the greatest number of nonzero entries in a column. This guarantees finding the



The Graph of the Circular Ones Problem of Figure 8.2

Figure 8.5

shortest path around the circle. This is because the solution must contain an arc which begins at one of the K nodes considered, since no arc is longer than K + 1 nodes.

Algorithm 8.2 may be interpreted in a similar manner. If (P) is considered as a shortest path problem on a circle, find the node I with the least number of arcs, say K, starting at that node or passing over that node. Then solve the shortest path problem starting and ending at node I, K times. Each time set one (a different one) of the K arcs passing over or starting at node I equal to one, and the other K - 1 equal to zero.

8.4 Networks on a Circle

These intepretations extend to thc general circular ones problem (P)

$$
\begin{aligned}
(P) \qquad & \text{Min} \quad CX \\
& \text{st} \quad AX = b \\
& \qquad X \ \text{Integer, Nonnegative}
\end{aligned}
$$

Now the problem is one of a network on a circle. Consider the example in Figure 8.1. Let I be the row of the A matrix with minimum b_i. Then apply transformation Q_I to A. See the example in Figure 8.6. Now repeat the first constraint as an M + 1st constraint to the problem and apply transformation T of Chapter 2 to the problem (see Figures 8.7 and 8.8). The problem has been transformed to a network flow problem with one side constraint $a_I X = b_I$, where a_I is the Ith row of A.

$$
\begin{aligned}
(P') \qquad & \text{Min} \quad CX \\
& \text{st} \quad a_I X = b_I \\
& \qquad A^* X = b^* \\
& \qquad\quad X \ \text{Integer, Nonnegative}
\end{aligned}
$$

Interpreting (P') in our circular network model, we see we have a normal network problem with the additional constraint that the amount of flow passing through OR PAST node 1 must equal b_I.

Formulation (P') suggests how easy it might be to solve the circular ones problem using Algorithm 3.1. There is only one complicating constraint in (P'). Therefore, we form the Langrangean relaxation

$$\text{Min} \quad \sum_{j=1}^{7} c_j X_j$$

st	X_1	X_2	X_3	X_4	X_5	X_6	X_7		RHS
4 am - 6 am	0	1	1	0	0	0	0	=	1
6 am - 8 am	0	1	1	0	0	0	0	=	1
8 am - 10 am	0	1	0	0	1	0	0	=	2
10 am - 12 m	0	1	0	0	1	0	0	=	1
12 m - 2 pm	0	0	0	0	1	0	1	=	2
2 pm - 4 pm	0	0	0	0	1	0	1	=	1
4 pm - 6 pm	1	0	0	0	0	0	1	=	1
6 pm - 8 pm	1	0	0	0	0	0	1	=	1
8 pm - 10 pm	1	0	1	0	0	0	0	=	1
10 pm - 12 pm	1	0	1	1	0	0	0	=	2
12 pm - 2 am	0	0	1	1	0	1	0	=	3
2 am - 4 am	0	1	1	1	0	0	0	=	2

$$X_j \geq 0, \text{ Integer} \quad j = 1,2,\ldots,7$$

The Circular Ones Example of Figure 8.1 Transformed by Q_3

Figure 8.6

$$\text{Min} \quad \sum_{j=1}^{7} c_j X_j$$

st	X_1	X_2	X_3	X_4	X_5	X_6	X_7		RHS
4 am - 6 am	0	1	1	0	0	0	0	=	1
6 am - 8 am	0	1	1	0	0	0	0	=	1
8 am - 10 am	0	1	0	0	1	0	0	=	2
10 am - 12 m	0	1	0	0	1	0	0	=	1
12 m - 2 pm	0	0	0	0	1	0	1	=	2
2 pm - 4 pm	0	0	0	0	1	0	1	=	1
4 pm - 6 pm	1	0	0	0	0	0	1	=	1
6 pm - 8 pm	1	0	0	0	0	0	1	=	1
8 pm - 10 pm	1	0	1	0	0	0	0	=	1
10 pm - 12 pm	1	0	1	1	0	0	0	=	2
12 pm - 2 am	0	0	1	1	0	1	0	=	3
2 am - 4 am	0	1	1	1	0	0	0	=	2
4 am - 6 am	0	1	1	0	0	0	0	=	1

$$X_j \geq 0, \text{ Integer} \qquad j = 1,2,\ldots,7$$

The Circular Ones Example of Figure 8.6 with the First Constraint Repeated as a Thirteenth Constraint

Figure 8.7

$$\text{Min} \quad \sum_{j=1}^{7} C_j X_j$$

st	X_1	X_2	X_3	X_4	X_5	X_6	X_7		RHS
4 am - 6 am	0	1	1	0	0	0	0	=	1
6 am - 8 am	0	0	0	0	0	0	0	=	0
8 am - 10 am	0	0	-1	0	1	0	0	=	1
10 am - 12 m	0	0	0	0	0	0	0	=	-1
12 m - 2 pm	0	-1	0	0	0	0	1	=	1
2 pm - 4 pm	0	0	0	0	0	0	0	=	-1
4 pm - 6 pm	1	0	0	0	-1	0	0	=	0
6 pm - 8 pm	0	0	0	0	0	0	0	=	0
8 pm - 10 pm	0	0	1	0	0	0	-1	=	0
10 pm - 12 pm	0	0	0	1	0	0	0	=	1
12 pm - 2 am	-1	0	0	0	0	1	0	=	1
2 am - 4 am	0	1	0	0	0	-1	0	=	-1
4 am - 6 am	0	0	0	-1	0	0	0	=	-1

$$X_j \geq 0, \text{ Integer} \qquad j = 1,2,\ldots,7$$

The Circular Ones Example of Figure 8.7 Transformed by T_{13}

Figure 8.8

$$W(U) = -Ub_I + \underset{\text{st}\quad X \text{ in } S}{\text{Min}}\ (C + Ua_I)X$$

where S is the set of solutions to the network flow problem defined by A*X = b*. W(U) is a concave piecewise linear function and U is a scalar. Consequently, the subgradient optimization reduces to a simple line search for a concave function.

CHAPTER 9

THE K DUTY PERIOD SCHEDULING PROBLEM

In this chapter we consider the K duty period scheduling problem. After describing the problem in Section 9.1, we show in the next section that the Lagrangean relaxation is once again a network flow problem. In Section 9.3 we demonstrate that subgradient optimization can be used to tighten the relaxation. Finally, we interpret the iterative process of the subgradient optimization.

9.1 The K Duty Period Scheduling Problem

Although thus far we have dealt exclusively with the two duty period scheduling problem, it is quite possible to extend our development to the more general K duty period scheduling problem where K = 1,2,3,.... The formulation would be

$$(P) \quad \text{Min} \quad CX$$
$$\text{st} \quad AX = b$$
$$X \text{ Integer, Nonnegative}$$

where each column of A contains at most K segments of ones, as defined in Chapter 2, the rest of the entries being zero.

The first consideration is whether or not this is a realistic problem. There are a number of possible applications. The first is simply the two duty period scheduling problem on a continuous twenty-four hour operation. This is analogous to the circular ones problem discussed in the previous chapter. For example, consider the problem of assigning drivers to buses which run 24 hours a day. It is not possible to formulate this problem as a two duty period scheduling problem so long as, for each row of the constraint matrix, there exists at least one duty period covering that row but not starting in it.

Suppose we are concerned with Bus A. Any 24 hour (or other) period need not have exactly two segments in each column in the constraint matrix. See Figure 9.1.

$$\text{Min} \quad \sum_{j=1}^{6} C_j X_j$$

st	X_1	X_2	X_3	X_4	X_5	X_6		RHS
12 pm - 2 am	0	1	1	0	1	1	=	2
2 am - 4 am	0	1	0	1	0	0	=	2
4 am - 6 am	0	0	1	1	0	0	=	1
6 am - 8 am	1	1	1	0	0	0	=	3
8 am - 10 am	1	1	0	1	0	0	=	2
10 am - 12 m	1	0	0	1	0	0	=	1
12 m - 2 pm	0	0	0	1	1	0	=	2
2 pm - 4 pm	0	0	0	0	1	0	=	1
4 pm - 6 pm	0	0	0	0	1	1	=	2
6 pm - 8 pm	0	0	0	0	0	1	=	1
8 pm - 10 pm	0	0	0	0	0	1	=	1
10 pm - 12 pm	0	0	1	0	1	1	=	3

$$X_j \geq 0, \quad \text{Integer} \quad j = 1,2,\ldots,6$$

An Example of the Circular Ones Problem with Two Continuous Duty Periods for Each Worker

Figure 9.1

Whereas possible driver schedules correspond to one or two segments (e.g. X_1, X_2) others might inevitably lead to three segments (e.g. X_3). Due to the wrap around nature of the continuous schedule a duty period might begin near the end of the column (e.g. 10 pm) and terminate near the beginning of the column (e.g. 2 am). This situation leads to three segments in the column. Consequently, the three duty period scheduling problem corresponds to scheduling personnel, each having at most two duty periods in a given time period (a day), where the job must be performed continuously from day to day.

It is not difficult to consider examples of the K duty period scheduling problem where K is greater than 3. Such a problem corresponds to scheduling personnel to serve at most K duty periods in a given planning horizon, for example five duty periods in seven days. Such a problem would be a K duty period scheduling problem in the case of linear time or a K + 1 duty period scheduling problem in the case of cyclic time.

An interesting extension of this idea is that the general M X N set partitioning problem can be viewed as an $[(M+1)/2]$ duty period scheduling problem, where

[(M+1)/2] is the largest integer less than or equal to (M+1)/2. This is simply because [(M+1)/2] is the greatest number of segments possible in a column with M rows.

9.2 The Relaxation

Consider the K duty period scheduling problem (P)

$$
\begin{aligned}
(P) \quad \text{Min} \quad & CX \\
\text{st} \quad & AX = b \\
& X \text{ Integer, Nonnegative}
\end{aligned}
$$

where each column contains at most K segments. Our relaxation will be to decouple the K segments in each column to give K new columns each with one segment. The relaxation is then a network flow problem. To create a problem equivalent to (P) one must add side constraints forcing the K variables replacing each original variable to be equal. We have already proved in Proposition 2.1 that (P') is equivalent to (P).

$$(P') \qquad \text{Min} \quad \sum_{j=1}^{N} \sum_{k=1}^{k_j} C_j^k X_j^k$$

$$\text{st} \quad \sum_{j=1}^{N} \sum_{k=1}^{k_j} A_j^k X_j^k = b$$

$$X_j^k - X_j^{k+1} = 0 \qquad j = 1,\ldots,N; \quad k = 1,\ldots,k_j - 1$$

$$X_j^k \geq 0, \text{ Integer} \qquad j = 1,\ldots,N; \quad k = 1,\ldots,k_j$$

where $\sum_{k=1}^{k_j} C_j^k = C_j, \quad j = 1,\ldots,N; \quad k_j \leq K.$

A column containing K segments in the original problem (P) will correspond to K arcs in the relaxation. If all of these partner arcs (corresponding to the same original column) have the same value in the relaxation, then there is, in essence, no relaxation. The more the partner arcs' values differ, the looser the relaxation will be. In this sense, we can say that the larger K is, the weaker the network flow relaxation will be. This is simply because the number of arcs has been increased.

9.3 Tightening the Relaxation

Once again it would be possible to use subgradient optimization to tighten the relaxation. Consider the K duty period scheduling problem (P') which may be rewritten as (P")

$$(P") \qquad \underset{X \text{ in } S}{\text{Min}} \sum_{j=1}^{N} \sum_{k=1}^{k_j} C_j^k X_j^k$$

$$\text{st} \quad X_j^k - X_j^{k+1} = 0 \qquad j = 1,\ldots,N$$

$$k = 1,\ldots,k_j - 1$$

where S is the set of solutions to the network flow problem given by the constraints

$$\sum_{j=1}^{N} \sum_{k=1}^{k_j} A_j^k X_j^k = b$$

Then the Langrangean relaxation of (P") is

$$W(U) = \underset{X \text{ in } S}{\text{Min}} \sum_{j=1}^{N} \sum_{k=1}^{k_j} C_j^k X_j^k + \sum_{j=1}^{N} \sum_{k=1}^{k_j - 1} U_j^k (X_j^k - X_j^{k+1})$$

$$= \underset{X \text{ in } S}{\text{Min}} \sum_{j=1}^{N} \sum_{k=1}^{k_j} (C_j^k + U_j^k - U_j^{k-1}) X_j^k$$

where $U_j^o \equiv 0$, $U_j^{k_j} \equiv 0$

and $U = (U_1^1, \ldots, U_1^{k_1}, U_2^1, \ldots, U_N^{k_N})$

$X = (X_1^1, \ldots, X_1^{k_1}, X_2^1, \ldots, X_N^{k_N})$

Since $W(U)$ has the integrality property, $\max_U W(U)$ will be equal to the linear programming relaxation of (P).

Consider the function $W(U)$. This is a piecewise linear concave function, since it is the pointwise minimum of a family of linear functions of U. Suppose $\hat{X} = (\hat{X}_1^1, \ldots, \hat{X}_1^{k_1}, \hat{X}_2^1, \ldots, \hat{X}_N^{k_N})$ is optimal for $W(\hat{U})$. Then as we have shown in Chapter 4, Section 4.1, $(\hat{X}^k - \hat{X}^{k+1})$ is a subgradient of W at $\hat{U}$, where

$$\hat{X}^k - \hat{X}^{k+1} = (\hat{X}_1^1 - \hat{X}_1^2, \ldots, \hat{X}_1^{k_1-1} - \hat{X}_1^{k_1}, \ldots, \hat{X}_N^{k_N-1} - \hat{X}_N^{k_N})$$

Consequently, we may again use subgradient optimization to tighten our relaxation. Let $\hat{W}$ be the target value. Then at iteration t + 1, we define

$$U^{t+1} = U^t + d^t[\hat{W} - W(U^t)]s^t / |s^t|^2$$

where $s^t \equiv (\hat{X}^k - \hat{X}^{k+1})^t$

$\equiv (\hat{X}^k - \hat{X}^{k+1})$ at iteration t

where d^t satisfies $0 < d^t \leq 2$, $d^t \to 0$,

and $\sum_{t=1}^{\infty} d^t = \infty$.

9.4 Interpreting the Subgradient Optimization Iteration

To interpret the iterative step defining the multipliers at iteration t + 1, let us develop a little terminology. Each variable X_j in our original problem was decoupled into k_j variables $(X_j^1, X_j^2, X_j^3, \ldots, X_j^{k_j})$. For each variable X_j^k, define its PARTNERS to be its PREDECESSOR X_j^{k-1}, and its SUCCESSOR X_j^{k+1}, where they exist (i.e. $k \neq 1, k_j$). Define the ASSOCIATES of X_j^k to be X_j^i, for all $i \neq k$, $1 \leq i \leq k_j$. Now, at iteration t + 1:

$$(C_j^k)^{t+1} = C_j^k + (U_j^k)^{t+1} - (U_j^{k-1})^{t+1}$$

Case 1: $1 < k < k_j$

$$
\begin{aligned}
(C_j^k)^{t+1} &= C_j^k + (U_j^k)^t + d^t[\hat{W} - W(U^t)](s_j^k)^t / |s^t|^2 \\
&\quad - (U_j^{k-1})^t - d^t[\hat{W} - W(U^t)](s_j^{k-1})^t / |s^t|^2 \\
&= (C_j^k)^t + d^t[\hat{W} - W(U^t)][(s_j^k)^t - (s_j^{k-1})^t] / |s^t|^2 \\
&= (C_j^k)^t + d^t[\hat{W} - W(U^t)][(\hat{x}_j^k)^t - (\hat{x}_j^{k+1})^t - \\
&\qquad (\hat{x}_j^{k-1})^t + (\hat{x}_j^k)^t] / |s^t|^2 \\
&= (C_j^k)^t + d^t[\hat{W} - W(U^t)][-(\hat{x}_j^{k-1})^t + 2(\hat{x}_j^k)^t - \\
&\qquad (\hat{x}_j^{k+1})^t] / |s^t|^2
\end{aligned}
$$

Case 2: $k = 1$

$$
\begin{aligned}
(C_j^k)^{t+1} &= C_j^k + (U_j^k)^{t+1} \\
&= C_j^k + (U_j^k)^t + d^t[\hat{W} - W(U^t)](s_j^k)^t / |s^t|^2 \\
&= (C_j^1)^t + d^t[\hat{W} - W(U^t)][(x_j^1)^t - (x_j^2)^t] / |s^t|^2
\end{aligned}
$$

Case 3: $k = k_j$

$$
\begin{aligned}
(C_j^k)^{t+1} &= C_j^k - (U_j^{k-1})^{t+1} \\
&= C_j^k - (U_j^{k-1})^t - d^t[\hat{W} - W(U^t)](s_j^{k-1})^t / |s^t|^2 \\
&= (C_j^k)^t - d^t[\hat{W} - W(U^t)][(\hat{x}_j^{k_j-1})^t - (\hat{x}_j^{k_j})^t] / |s^t|^2
\end{aligned}
$$

Note for the case $k_j = 2$, Case 3 above becomes

Case 3*: $k = k_j = 2$

$$(C_j^2)^{t+1} = (C_j^2)^t - d^t[\hat{W} - W(U^t)][(X_j^1)^t - (X_j^2)^t]/|s^t|^2$$

and Cases 2 and 3* define the iterative formula developed in Chapter 4 for C^{t+1} for the two duty period scheduling problem, where $X_j^1 = Y_j$ and $X_j^2 = Z_j$.

Let us examine the general formula given in Case 1 above. Recall

$$s^t = (s_j^k)^t \qquad j = 1,\ldots,N; \quad k = 1,\ldots,k_j$$

Then

$$|s^t|^2 = \sum_{j=1}^{N} \sum_{k=1}^{k_j-1} (s_j^k)^2$$

$$= \sum_{j=1}^{N} \sum_{k=1}^{k_j-1} (X_j^k - X_j^{k+1})^2$$

So $|s^t|^2$ is equal to the number of times a decoupled arc is different than its successor. Note that in Case 1 above

$$-2 \leq -(\hat{X}_j^{k-1})^t + 2(\hat{X}_j^k)^t - (\hat{X}_j^{k+1})^t \leq 2$$

Let P^t be the penalty at iteration t + 1

$$P^t \equiv [\hat{W} - W(U^t)] / |s^t|^2$$

Then

$(C_j^k)^{t+1} = (C_j^k)^t + 2P^t$ if X_j^k is in the current solution but neitner its successor nor its predecessor is

$(C_j^k)^{t+1} = (C_j^k)^t + P^t$ if X_j^k is in the current solution and either its successor or predecessor (but not both) is also in

$(C_j^k)^{t+1} = (C_j^k)^t$ if X_j^k and its successor and predecessor are either all in or all not in the current solution

$(C_j^k)^{t+1} = (C_j^k)^t - p^t$ if X_j^k is not in the current solution but either its successor or predecessor (but not both) is in

$(C_j^k)^{t+1} = (C_j^k)^t - 2p^t$ if X_j^k is not in the current solution but both its successor and predecessor are in.

9.5 Summary

The K duty period scheduling problem may be handled in a way analogous to the two duty period scheduling problem. By decoupling the segments in each column the problem can be reformulated as a network flow problem with side constraints. In the case of a set partitioning problem the relaxation becomes an acyclic shortest path problem. When an arc is chosen to be in the solution to the shortest path relaxation, all of its associates are also chosen. It is then possible to eliminate all arcs (and their associates) which conflict with these chosen arcs. Similarly, using the backward-forward iterative solution procedure for the

shortest path problem, it is possible to eliminate arcs which cannot participate in a solution better than the best known solution. Of course, when an arc is eliminated, its associates are also eliminated.

CHAPTER 10

GENERAL METHODOLOGY

In this chapter we consider the general procedure of decoupling columns to form problems amenable to Lagrangean relaxation techniques. It is shown that the methodology is applicable to near block diagonal matrices.

10.1 Solution Procedure for the Two Duty Period Scheduling Problem

The previous chapters have been devoted to developing a solution procedure for the two duty period scheduling problem. In this chapter we try to examine the methodology that has been developed and see to what extent it may be generalized.

We began with the two duty period scheduling problem (P):

$$(P) \quad \text{Min} \quad CX$$
$$\text{st} \quad AX = b$$
$$X \text{ Integer, Nonnegative}$$

where each column of A contained exactly two segments of ones. As such, the problem could be handled only by general integer programming techniques. However, it was recognized that by decoupling the columns, the problem could be transformed into a network flow problem with side constraints. These side constraints were very simple, of the form $Y - Z = 0$.

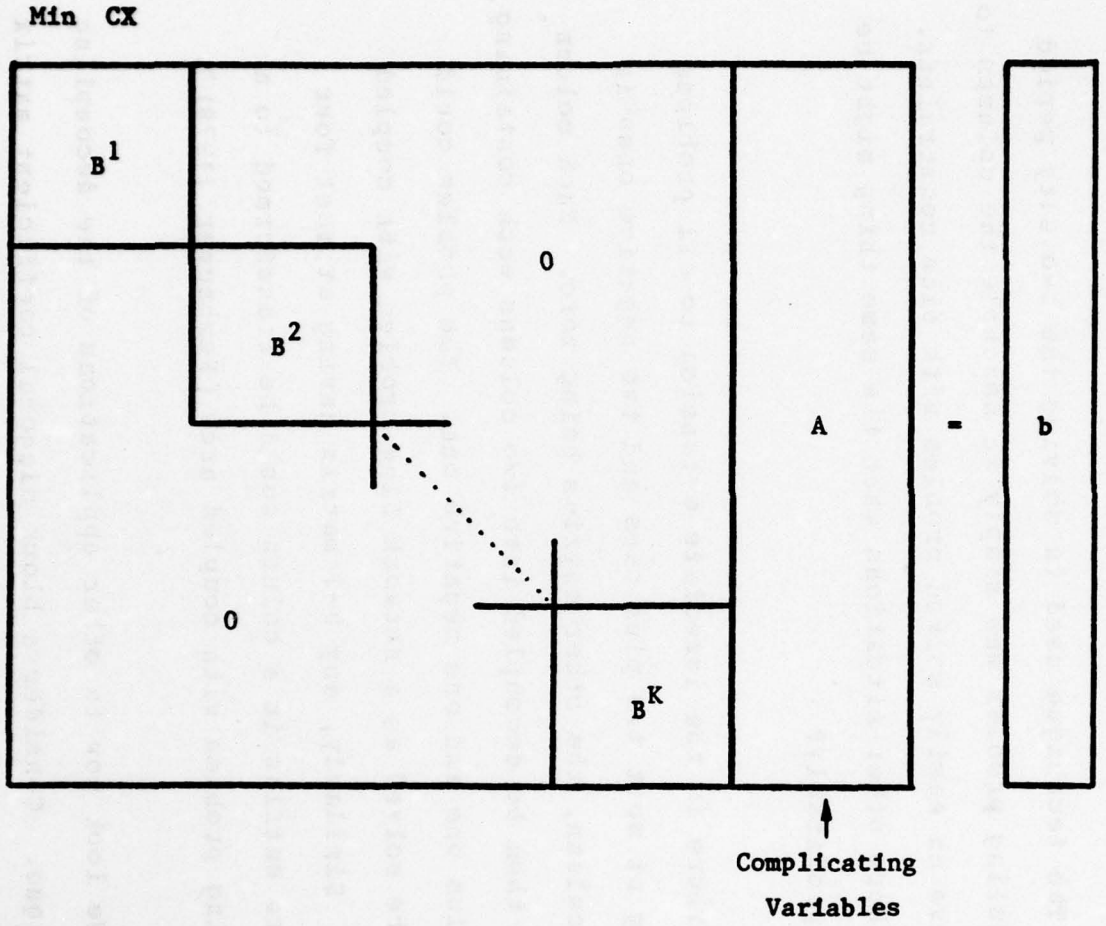
It was shown that by putting these side constraints into the objective function, a very easily evaluated Lagrangean could be formed. Moreover, due to the simple nature of the side constraints, the dual problem could be solved very efficiently by subgradient optimization techniques. The computation involved at each iteration was very slight compared to solving the linear programming relaxation of the original two duty period scheduling problem.

10.2 Broader Applications of the Methodology Developed to Solve the Two Duty Period Scheduling Problem

The technique used in solving the two duty period scheduling problem was simply to decouple the columns to achieve an easily solved problem with side constraints. Are there other situations when the same thing might be done profitably?

There is the immediate extension to all problems having at most two plus ones and two negative ones in each column, the other entries being zero. Each column could then be decoupled into two columns each containing one plus one and one negative one. The problem could then be solved as a network flow problem with coupled arcs. Similarly, any 0-1 matrix having at most four nonzero entries in a column could be transformed to a matching problem with coupled arcs [Nemhauser (1978)].

We look now to other applications of the decoupling technique. Consider a block diagonal coefficient matrix with K blocks and with complicating variables (see Figure 10.1). Benders Decomposition is a well known

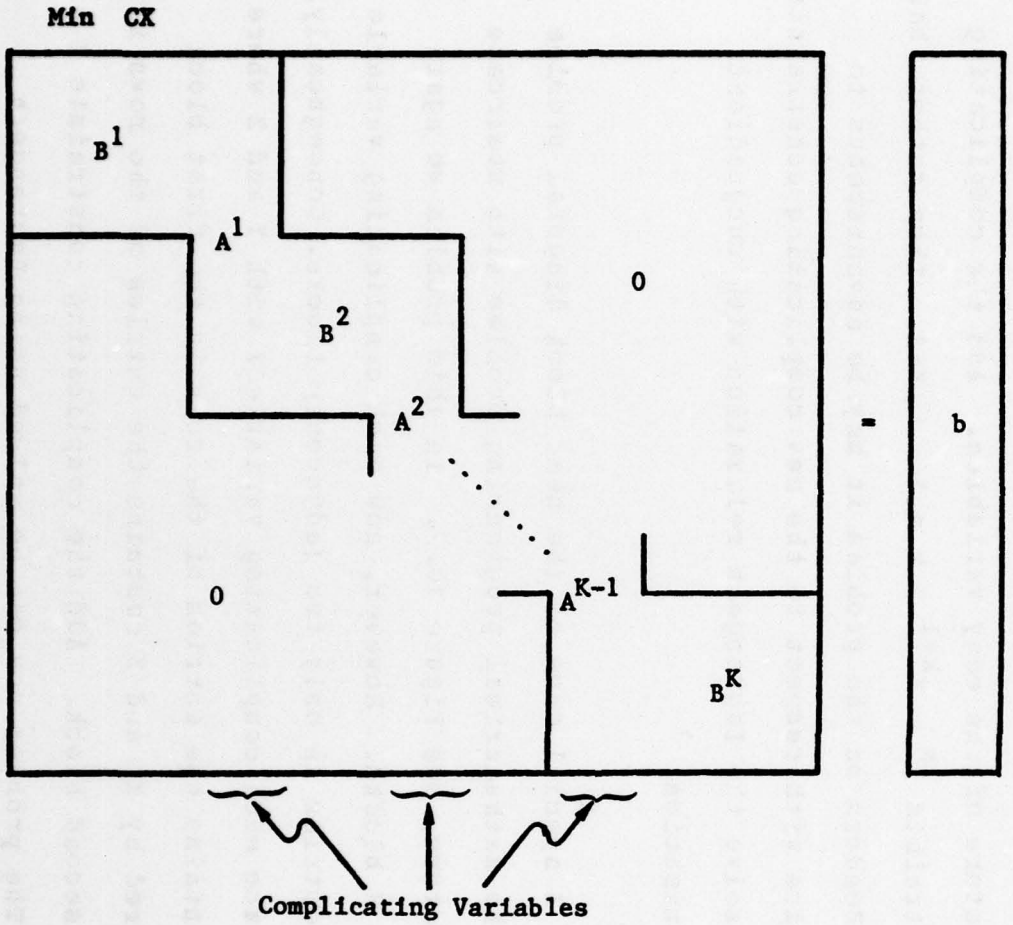


An Example of a Near Block Diagonal Problem with Complicating Variables

Figure 10.1

technique for handling this particular problem. An alternative method would be to decouple the complicating variables. Replace each complicating variable X with K variables X^k, k = 1,...,K, where X^k contains the entries of the rows in the kth block in the block diagonal structure of the easy variables. Add the complicating constraints $X^k = X^{k+1}$, k = 1,...,K-1. Then rather than use Benders on the problem it may be advantageous to dualize with respect to the new complicating constraints and solve the Lagrangean relaxation with subgradient optimization.

A special case of the near block diagonal problem is the mathematical programming problem with staircase structure, see Figure 10.2. In this problem we again have K blocks. However, now each complicating variable has entries in only two (adjacent) blocks. Consequently, replace each complicating variable X with Y and Z where Y contains the entries of the rows in the first block covered by X, and Z contains the entries of the rows in the second block. Add the complicating constraints Y = Z. The problem now can be solved using Lagrangean relaxation and subgradient optimization.



A Problem with Staircase Structure

Figure 10.2

A similar situation arises for near block diagonal problems with complicating constraints, see Figure 10.3. Here we consider the simplest case of only two blocks. Consider a complicating constraint

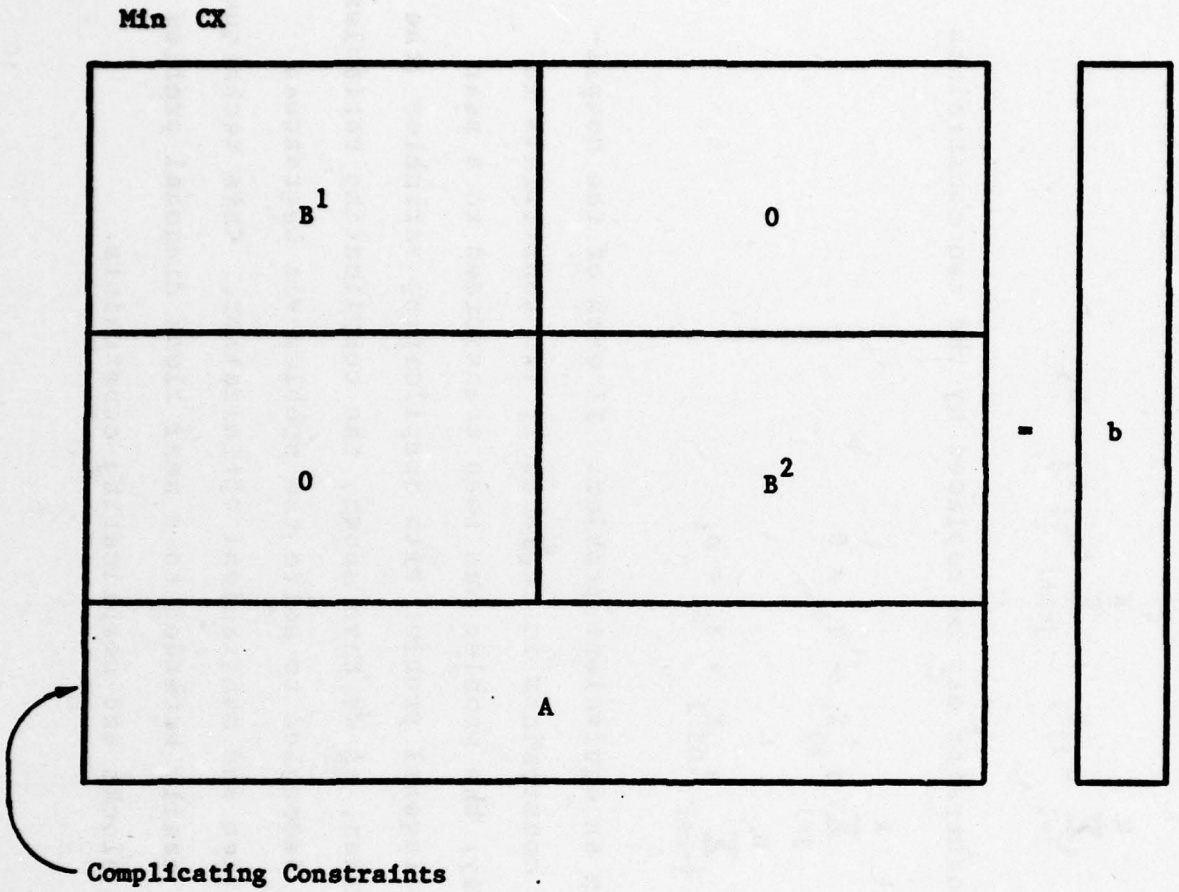
$$\sum_{j=1}^{k} a_{ij}X_j + \sum_{j=k+1}^{N} a_{ij}X_j = b_i$$

This constraint may be replaced by the two constraints

$$\sum_{j=1}^{k} a_{ij}X_j - Y_i = 0$$

$$\sum_{j=k+1}^{N} a_{ij}X_j + Y_i = b_i$$

to form an equivalent problem. If each of the complicating constraints is replaced by two constraints in this way, the problem has been transformed to a near block diagonal problem with complicating variables (the Y_i). Then, as we have shown, the complicating variables may be decoupled to solve the problem via Lagrangean relaxation and subgradient optimization. This technique may be easily extended to a near block diagonal problem with K blocks and complicating constraints.



A Near Block Diagonal Problem with Complicating Constraints

Figure 10.3

BIBLIOGRAPHY

1. Agmon, S., "The Relaxation Method for Linear Inequalities," Canadian journal of Mathematics, Vol. 6, No. 3, (1954), pp. 382-392.

2. Assad, A.A., "Multicommodity Network Flows - Computational Experience," Working Paper OR 058-76, Operations Research Center, Massachusetts Institute of Technology, (October 1976).

3. Baker, K.R., "Scheduling a Full-Time Workforce to Meet Cyclic Staffing Requirements," Management Science, Vol. 20, No. 12, (August 1974), pp. 1561-1568.

4. Baker, K.R., "Workforce Allocation in Cyclical Scheduling Problems: Models and Applications," G.S.B.A. Paper No. 122, Graduate School of Business Administration, Duke University, (June 1975).

5. Bartholdi, J.J., and Ratliff, H.D., "Unnetworks, With Applications to Idle Time Scheduling," Research Report No. 77-4, Industrial and Systems Engineering Department, University of Florida, Gainesville, (April 1977).

6. Bartholdi, J.J., Orlin, J.B., and Ratliff, H.D., "Circular 1's and Cyclic Staffing," Research Report 77-11, Industrial and Systems Engineering Department, University of Florida, (1977).

7. Brownell, W.S., and Lowerre, J.M., "Scheduling of Work Forces Required in Continuous Operations Under Alternative Labor Policies," Management Science, Vol. 22, No. 5, (January 1976), pp. 597-605.

8. Cornuejols, G., Fisher, M.L., and Nemhauser, G.L., "Location of Bank Accounts to Optimize Float: An Analytic Study of Exact and Approximate Algorithms," Management Science, Vol. 23, No. 8, (April 1977), pp. 789-810.

9. Denardo, E.V., and Fox, B.L., "Shortest Route Methods: 1. Reaching, Pruning, and Buckets," Publication No. 263, Departement d'Informatique et de Recherche Operationelle, Universite de Montreal, Canada, (September 1977).

10. Etcheberry, J., "The Set Representation Problem: A New Implicit Enumeration Algorithm," Pub. No. 76/18/C, Centro de Planeamiento - Departamento de Industrias, Universidad de Chile - Sede Occidente, (September 1976).

11. Everett, H., "Generalized Lagrange Multiplier Method for Solving Problems of Optimum Allocation of Resources," Operations Research, Vol. 11, No. 3, (May-June 1963), pp. 399-417.

12. Fisher, M.L., "Optimal Solution of Scheduling Problems Using Lagrange Multipliers, Part I," Operations Research, Vol. 21, No. 5, (September-October 1973), pp. 1114-1127.

13. Fisher, M.L., "Optimal Solution of Scheduling Problems Using Lagrange Multipliers, Part II," Symposium on Theory of Scheduling and its Applications, North Carolina State University, Raleigh, (1972).

14. Fisher, M.L., Northup, W.D., and Shapiro, J.F., "Using Duality to Solve Discrete Optimization Problems: Theory and Computational Experience," Mathematical Programming Study 3, (1975), pp. 56-94.

15. Garfinkel, R.S., and Nemhauser, G.L., Integer Programming, Wiley & Sons, New York, (1972).

16. Geoffrion, A.M., "Duality in Nonlinear Programming: A Simplified Applications-Oriented Development," Society for Industrial and Applied Mathematics Review, Vol. 13, No. 1, (January 1971), pp. 1-37.

17. Geoffrion, A., "Lagrangean Relaxation for Integer Programming," Mathematical Programming Study 2, (1974), pp. 82-114.

18. Geoffrion, A.M., and Marsten, R.E., "Integer Programming Algorithms: A Framework and State-of-the-Art Survey," Management Science, Vol. 18, No. 9, (May 1972), pp.465-491.

19. Glover, F., "Integer Programming over a Finite Additive Group," SIAM Journal of Control, Vol. 7, No. 2, (May 1967), pp. 213-231.

20. Glover, F., "Surrogate Constraints," Operations Research, Vol. 16, No. 4, (July-August 1968), pp. 741-749.

21. Glover, F., "Surrogate Constraint Duality in Mathematical Programming," Operations Research, Vol. 23, No. 3, (May-June 1975), pp. 434-451.

22. Glover, F., and Mulvey, J.M., "Network-Related Scheduling Models for Problems with Quasi-Adjacency and Block Adjacency Structure," Working Paper HBS 76-3, Graduate School of Business Administration, Harvard University, (1976).

23. Goffin, J.L., "On the Finite Convergence of the Relaxation Method ORC 71-36, University of California, Berkeley, (1971).

24. Goffin, J.L., "On Convergence Rates of Subgradient Optimization Methods," Working Paper No. 76-34, Faculty of Management, McGill University, (August 1976), (to appear in Mathematical Programming).

25. Greenberg, H.J., and Pierskalla, W.P., "Surrogate Mathematical Programs," Operations Research, Vol. 18, No. 5, (September-October 1970), pp. 924-939.

26. Held, M., and Karp, R.M., "The Traveling-Salesman Problem and Minimum Spanning Trees," Operations Research, Vol. 18, No. 6, (November-December 1970), pp. 1138-1162.

27. Held, M., and Karp, R.M., "The Traveling-Salesman Problem and Minimum Spanning Trees: Part II," Mathematical Programming Study 1, (1971), pp. 6-25.

28. Held, M., Wolfe, P., and Crowder, H., "Validation of Subgradient Optimization," Mathematical Programming Study 6, (1974), pp. 62-88.

29. Karwan, M.H., and Rardin, R.L., "Surrogate Dual Multiplier Search Procedures in Integer Programming," Report Series No. J-76-33, School of Industrial and Systems Engineering, Georgia Institute of Technology, (October 1976).

30. Kennington, J.L. and Shalaby, M., "An Effective Subgradient Procedure for Minimal Cost Multicommodity Flow Problems," Management Science, Vol. 23, No. 9, (May 1977), pp. 994-1004.

31. Koljonen, S., and Tamminen, M., "Bus Crew Scheduling at Helsinki City Transport," presented at the Nordic Operations Analysis Conference (NOAK 77), (October 1977).

32. Magnanti, T.L., "Optimization for Sparse Systems," in Sparse Matrix Computations (J.R. Bunch and D.J. Rose, editors), Academic Press, New York, (1976), pp. 147-176.

33. Magnanti, T.L. and Golden, B.L., "Transportation Planning: Network Models and their Implementation," Technical Report No. 143, Operations Research Center, Massachusetts Institute of Technology, (January 1978), (to appear in Studies in Operations Management; A.C. Hax, editor).

34. Maier-Rothe, C., and Wolfe, H.B., "Cyclical Scheduling and Allocation of Nursing Staff," Socio-Economic Planning Sciences, Vol. 7, No. 5, (October 1973), pp. 471-487.

35. Marsten, R.E., "An Implicit Enumeration Algorithm for the Set Partitioning Problem with Side Constraints," Working Paper No. 181, Western Management Science Institute, University of California, Los Angeles, (October 1971).

36. Marsten, R.E., and Morin, T.L., "A Hybrid Approach to Discrete Mathematical Programming," Mathematical Programming, Vol. 14, No. 1, (January 1978), pp. 21-40.

37. Motzkin, T.S., and Schoenberg, I.J., "The Relaxation Method for Linear Inequalities," Canadian Journal of Mathematics, Vol. 6, No. 3, (1954), pp. 393-404.

38. Mulvey, J.M., "A Network Relaxation Approach for the Set Partitioning and Set Covering Models," Working Paper HBS 75-37, Graduate School of Business Administration, Harvard University, (1975).

39. Nemhauser, G.L., private communication, (March 1978).

40. Orlin, J., "Quick Optimal Weekly Scheduling with Two Consecutive Days Off," Technical Report 77-1, Department of Operations Research, Stanford University, (January 1977).

41. Poljak, B.T., "A General Method of Solving Extremum Problems," Soviet Mathematics Doklady, Vol. 8, No. 3, (May-June 1967), pp. 593-597.

42. Segal, M., "The Operator-Scheduling Problem: A Network-Flow Approach," Operations Research, Vol. 22, NO. 4, (July-August 1974), pp. 808-823.

43. Shapiro, J.F., "Dynamic Programming Algorithms for the Integer Programming Problem - I: The Integer Programming Problem Viewed as a Knapsack Type Problem," Operations Research, Vol. 16, No. 1, (January-February 1968), pp. 103-121.

44. Shapiro, J.F., and Wagner, H.M., "A Finite Renewal Algorithm for the Knapsack and Turnpike Models," Operations Research, Vol. 15, No. 2, (March-April 1967), pp. 319-341.

45. Tibrewala, R., Philippe, D., and Browne, J., "Optimal Scheduling of Two Consecutive Idle Periods," Management Science, Vol. 19, No. 1, (September 1972), pp. 71-75.

46. Tucker, A., "Matrix Characterization of Circular-Arc Graphs," Pacific Journal of Mathematics, Vol. 39, No. 2, (1971), pp. 535-545.

47. Veinott, A.F., and Wagner, H.M., "Optimal Capacity Scheduling - I," Operations Research, Vol. 10, No. 4, (1962), pp. 518-532.